AFRL-IF-RS-TR-2004-300
Final Technical Report
October 2004

POWER AWARE MULTIPROCESSOR ARCHITECTURE (PAMA)

University of Southern California/ISI

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J875

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2004-300 has been reviewed and is approved for publication

APPROVED: /s/
MARTIN J. WALTER
Project Engineer

FOR THE DIRECTOR: /s/
JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 074-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** October 2004	**3. REPORT TYPE AND DATES COVERED** **FINAL Jun 00 – Dec 03**	
4. TITLE AND SUBTITLE POWER AWARE MULTIPROCESSOR ARCHITECTURE (PAMA)			**5. FUNDING NUMBERS** G - F30602-00-2-0548 PE - 62301E PR - J875 TA - PA WU - MA
6. AUTHOR(S) Stephen Crago			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Southern California/ISI 4676 Admiralty Way, Suite 1001 Marina Del Rey CA 90292-6695			**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL/IFTC 26 Electronic Parkway Rome NY 13441-4514; Defense Advanced Research Projects Agency 3701 North Fairfax Drive Arlington VA 22203-1714			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2004-300
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Martin J. Walter/IFTC/(315) 330-4102 Martin.Walter@rl.af.mil			
12a. DISTRIBUTION / AVAILABILITY STATEMENT *APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

Many embedded signal-processing applications require the computing performance of multiprocessors and have varying power constraints imposed by their operating environment. These performance and power constraints can vary dynamically as situations and environments change. High-performance applications often require multiprocessor hardware to achieve sufficient processing performance. However, the need for high performance does not preclude the relevance of power management, either to extend battery life or to mitigate heat dissipation problems. The Power Aware Multiprocessor Architecture (PAMA) project has developed a power-aware multiprocessor architecture and has investigated the application of power management techniques to a space-based remote sensing application. The PAMA project built three generations of prototypes and demonstrated significant power and energy savings from changing algorithms, system software, and hardware. The PAMA project also built an e-textile prototype for a beamforming demonstration.

14. SUBJECT TERMS Power management, multiprocessor architecture			**15. NUMBER OF PAGES** 55
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Table of Contents

List of Figures

List of Tables

I. Introduction

Many embedded signal-processing applications require the computing performance of multiprocessors and have varying power constraints imposed by their operating environment. These performance and power constraints can vary dynamically as situations and environments change. Such applications require different trade-offs and techniques than low-power applications with lower processing requirements in which the primary goal is the extension of battery life. High-performance applications often require multiprocessor hardware to achieve sufficient processing performance. However, the need for high performance does not preclude the relevance of power management, either to extend battery life or to mitigate heat dissipation problems.

The Power Aware Multiprocessor Architecture (PAMA) project has developed a power-aware multiprocessor architecture and software library and has investigated the application of power management techniques to a space-based remote sensing application. The power management techniques supported by PAMA at the different levels of an overall system architecture are shown in Figure 1.

1) Processor/Interconnect level

The PAMA project designed and built three generations of the PAMA platform, an embedded multiprocessor architecture. Detailed descriptions of those platforms are presented in Section II. The PAMA platforms provide the following hardware knobs for power management.

Dynamic number of processors: Each processor in PAMA can be put to sleep. Therefore, the number of processors can be varied as the power budget and application demands change.

Independently variable processor clock speeds: Each processor in PAMA can run at its own clock speed. The clock speeds can be varied as power availability changes and application demands change.

Variable Power Interconnect: The network bandwidth can be varied with power budget and application demands. The interconnection between the processors within a board is implemented with programmable logic. The network can be specialized for each application or over the course of a single application to best use power resources.

Heterogeneous Processing Elements: PAMA contains both general-purpose computing resources and adaptive computing elements. Tasks that are well suited at adaptive computing will require less power on an adaptive computing element than on a general-purpose processor. Since PAMA contains both types of resources, the application programmer or design tool can balance power consumption and programming effort.

Power Aware Memory Control. Power consumption of memory can be managed for a given application by allowing changes of memory modes and bus frequencies.

2) OS/Runtime System level

Underlying processor/interconnect-level parameters such as clock speed and the number of processing elements need to be controlled by power-aware runtime software. We modified stock Linux to manage the underlying power-aware resources so that frequency scaling and voltage scaling are provided by Linux. We also provide a multiprocessor communication protocol, MPI (message passing interface), on top of Linux. All the available power knobs are exposed to application programs through device drivers and APIs. Details are presented in Section II.

3) Software/Compiler level

We developed compile-time design synthesis techniques to synthesize power parameters for optimal usage of energy on the PAMA architecture. We also developed run-time scheduling techniques and a library to control power consumption dynamically. Detailed descriptions are presented in Section III.

4) Algorithm level

At the top level, we developed algorithmic power management knobs for the FORTE space-based remote sensing application [1]. We traded accuracy for power in the signal processing algorithms of the FORTE to provide adaptive power management. Our algorithm-level power management is discussed in detail in Section IV.

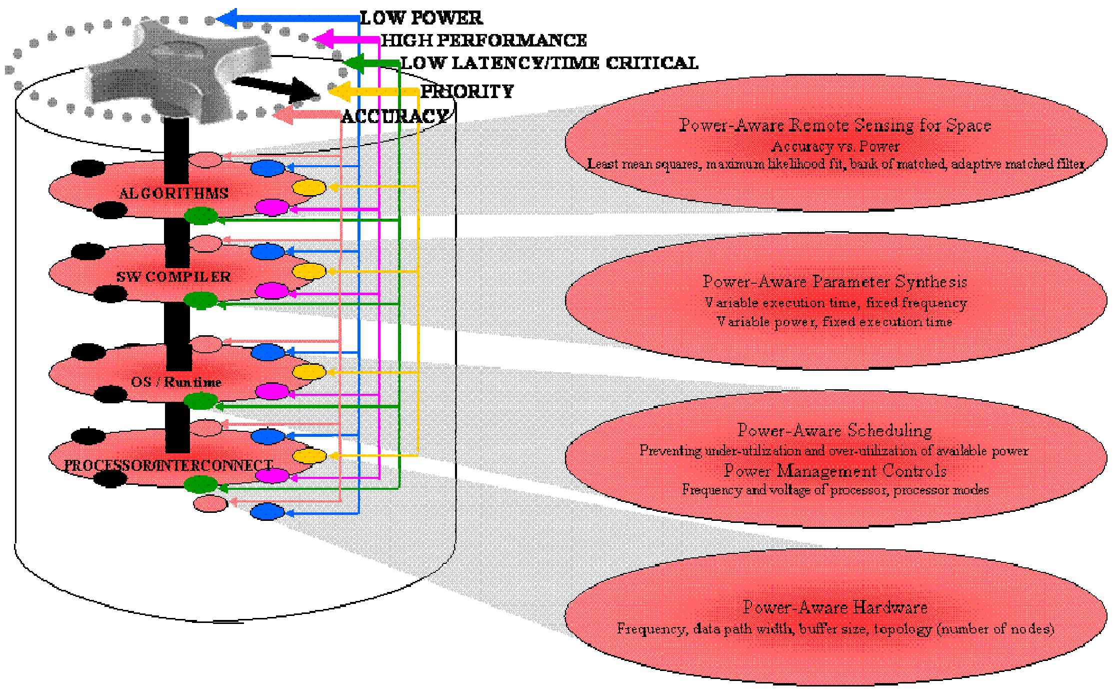

LOW POWER
HIGH PERFORMANCE
LOW LATENCY/TIME CRITICAL
PRIORITY
ACCURACY
ALGORITHMS
SW COMPILER
OS / Runtime
PROCESSOR/INTERCONNECT
Power-Aware Remote Sensing for Space
Accuracy vs. Power
Least mean squares, maximum likelihood fit, bank of matched, adaptive matched filter
Power-Aware Parameter Synthesis
Variable execution time, fixed frequency
Variable power, fixed execution time
Power-Aware Scheduling
Preventing under-utilization and over-utilization of available power
Power Management Controls
Frequency and voltage of processor, processor modes
Power-Aware Hardware
Frequency, data path width, buffer size, topology (number of nodes)

Figure 1. PAMA power knobs

II. PAMA Platforms

II.A. Hardware

II.A.1. SLIIC-QL Board

In the first phase of PAMA, we leveraged hardware designed under the SLIIC (System-Level Intelligent Intensive Computing) project to implement a baseline PAMA architecture that didn't require hardware development time. The SLIIC project was sponsored by DARPA ITO's Data Intensive Systems program. The SLIIC Quick Look (QL) board has all of the hardware components necessary to implement an initial version of the PAMA system. A block diagram of the SLIIC QL board is shown in Figure 2. The board is a PCI card that plugs into a standard PC system. The board contains eight Mitsubishi M32R/D chips, which are commercially available Processor-In-Memory (PIM) chips. A Xilinx XC4062 field-programmable gate array (FPGA) is used to implement the PCI interface and control registers. Two XC4085 FPGAs are used to implement the processor interconnection, control, and adaptive computing component. Table 1 shows M32/RD performance and power consumption relative to other processors.

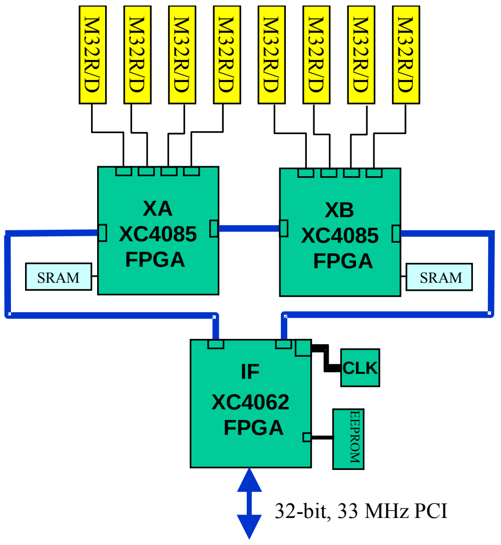


Figure 2. SLIIC QL board block diagram

Table 1. M32/RD performance/power comparison

Processor	Peak Performance	Typical Power	Performance/Power
DEC Alpha 21264	3.6 GIPS	72W	50 MIPS/W
Power PC 603	160 MIPS	3.0W	53 MIPS/W
M32R/D	62.9 MIPS	0.531W	113 MIPS/W

The PAMA project extended the SLIIC architecture to provide power management capability, which includes processor mode changes and operating frequency changes. The processors can be in either active, sleep, or stand-by mode [2]. In active mode, the full circuit is active (546 mW typical). In sleep mode, only the memory in the chip is working (393 mW typical). In stand-by mode, all circuits are stopped except the interrupt monitoring circuit (6.6 mW typical). Our firmware and software do not support sleep mode for the M32R/D because we did not use the M32R/D as a stand-alone memory chip. The board supports changing the number of processors and frequency. Possible frequencies are 20, 40, and 80 MHz. Limitations of the SLIIC QL board include the lack of hardware support voltage scaling and the absence of hardware support for floating point operations.

II.A.2. PAMA SH-4-Based Architecture

In the second phase of the PAMA project, we designed an enhanced PAMA architecture that builds on the experience gained on the original QL board. Feedback from Power Aware Computing and Communication (PAC/C) community and the experience of application-mapping on SLIIC QL board revealed the need to increase processing power. For the applications of interest, hardware support for floating point operations was required. Furthermore, it was important to provide a standard, operating system-based programming environment. Due to the small amount of memory on the M32R/D, no operating system (OS) or standard message passing protocol could be implemented on the SLIIC QL board. Providing a commercial off-the-shelf (COTS) OS such as Linux was desired to facilitate application development and portability of applications. Designing new PAMA hardware also provided the opportunity to update the programmable logic and to supply support for voltage scaling.

The hardware of the second phase of the PAMA was designed and built from scratch. We surveyed the technologies as is shown in Table 2 to find a processor technology with high MIPS/Watt. We chose the Hitachi SH-4 processor for the processor and the Xilinx Virtex II chip for programmable logic. The Hitachi SH-4 processor was used in the Sega DreamCast game console because of its high performance and power efficiency. Its peak floating point performance is 1.4 GFLOPS and a pre-existing Linux port for the SH-4 processor was available. Linux provided excellent programming support for our experimental platform. The upgrade to the Xilinx Virtex II field programmable gate array (FPGA) provided a more than ten times improvement in logic density over the 4000 series part used in the SLIIC QL board. In addition to the improvement of logic density, the power consumption per logic gate is also reduced.

Table 2. SH-4 efficiency comparison

Processor	MIPS (Peak)	Power (Watts)	MIPS/Watt	Comments
Imagine	40,000	2.5	7,000	Research chip
IRAM	3,200	2	1,600	Research chip
SuperH (SH-4)	1,400	0.9	1,556	Embedded processor
Crusoe 5400	1,300	1.5	867	Availability uncertain
TI ‘C6701	1,000	1.8	555	Digital signal processor
PPC G4	1,600	5	320	Embedded processor
SHARC 21160M	480	2	240	Digital signal processor
Mobile Pentium III	1,700	10	170	High power processor
MIPS 32300 (IDT)	133	1.5	89	Embedded processor

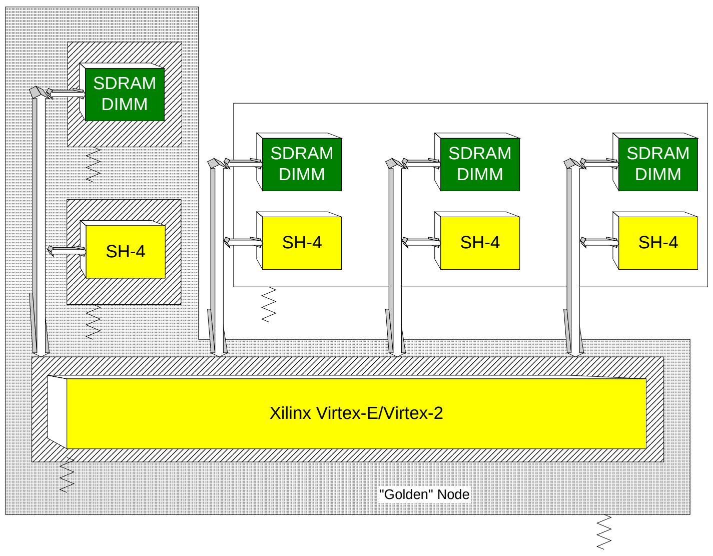
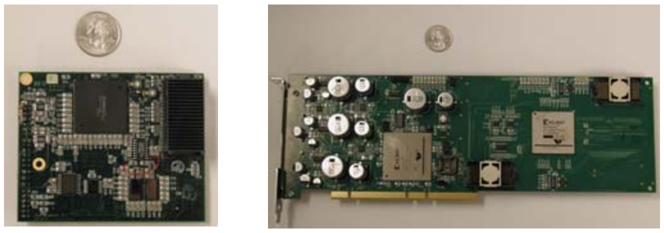


Figure 3. PAMA-SH-4 block diagram



(a) (b)

Figure 4. Picture of (a) SH-4 node card and (b) network card

The block diagram of the PAMA SH-4 architecture is shown in Figure 3. The memory module is a COTS laptop memory module, which is not only cheap but easily upgradeable and replaceable. One major design decision was to separate the processor boards and the network board. The major advantages of separating the processor board from the network board are: 1) the network card design is reusable with other CPU boards and 2) improved debugability. The connections between a processor board and a network board are done with a high-density COTS connector. A processor board contains one processor, one memory module, and other circuitry including a serial interface and a non-volatile memory. Separate power planes were designed for the processor, memory network, and other logic so that the power for each component group could be separately. The network board is designed as a PCI (Peripheral Component Interface) card, which carries up to four processor boards.

Table 3. Hardware support for power management for PAMA-SH

Power Knob	Hardware		Range
Clock Scaling	External Clock Synthesizer (EXT_CLK)		8 MHz ≤ EXT_CLK ≤ 33 MHz
	Internal PLL (1 PLL for CPU_CLK)	CPU (CPU_CLK)	{3/2, 2, 3, 6} x EXT_CLK, Maximum 200MHz
		Bus (BUS_CLK)	{¾, 1, 3/2, 2, 3} x EXT_CLK, Maximum 100MHz
Voltage Scaling	CPU only (CPU_VOLTAGE), Nominal voltage: 1.95 V.		1.35 V ≤ CPU_VOLTAGE ≤ 1.95 V, stabilization time = 100 us x (target voltage − original voltage) / 50 mV.
Operating Mode	CPU		{Full, Standby, Sleep}
	Memory		{Full, self-refresh} Self-refresh mode is allowed only when the processor is in "Standby" mode. "Full" mode is re-entered when an unmasked interrupt occurs.

The network board also has four dynamic programmable voltage regulators for four processor and four programmable clock synthesizers for four processor boards. A single Virtex II 6000 part provides both the network and control logic for power arrangement. The communication network is a five-point crossbar switch architecture. It is 64 bits wide and runs up to 30 MHz. The maximum bandwidth is 240 MB/s for each point to point communication. Pictures of the processor board and the network board are shown in Figure 4. The PAMA SH-4 architecture provides clock scaling, voltage scaling, and operating mode knobs for power management. The ranges of these power knobs and constraints are summarized in Table 3.

II.A.3. The PAMA PPC-Based Architecture

In the third phase of the PAMA project, we developed an enhanced PAMA architecture with the IBM PowerPC (PPC) 750FX microprocessor. The IBM 750FX was chosen because the PowerPC architecture is the most prevalent general-purpose processor architecture for space-based applications. The block diagram of the PAMA PowerPC-based architecture is shown in Figure 5. The PowerPC chip required the addition of a bridge chip that controls main memory. We designed the processor board with the IBM PowerPC 750FX, but reused the network board without modification, which demonstrates the flexibility of PAMA architecture. Firmware was ported to handle PowerPC-specific signals.

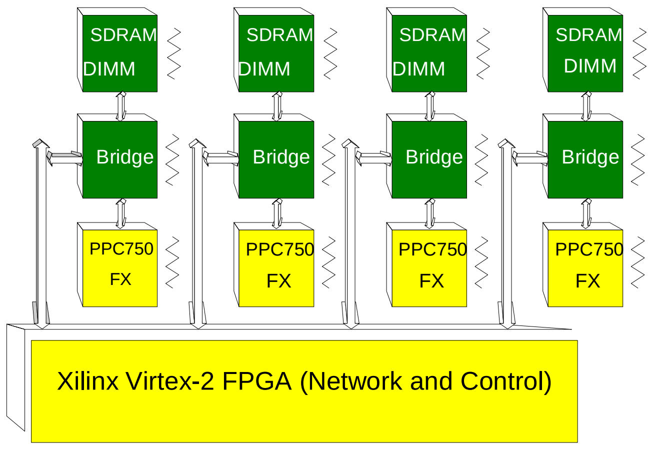


Figure 5. Block diagram of PAMA PPC architecture

Table 4. Power-knob ranges for the PAMA-PPC architecture

Power Knob	Hardware		Range
Clock Scaling	External Clock Synthesizer (EXT_CLK)		17 MHz ≤ EXT_CLK ≤ 20 MHz
	Internal PPL (2PLLs for CPU_CLK)	Bridge (Bridge_CLK)	3 x EXT_CLK
		CPU (CPU_CLK)	{1m3m3m5m,,,9.5,10,11,…,20} x CPU_CLK, Maximum 733 MHz, Minimum 180 MHz
		Bus (BUS_CLK)	Bridge_CLK, Maximum 100 MHz
Voltage Scaling	CPU only (CPU_VOLTAGE), Nominal voltage: 1.40 V		1.30 V < CPU_VOLTAGE < 1.4V, stabilization time = 100 us x (target voltage - original voltage) / 50 mV
Operating Mode	CPU		{Full, Doze, Nap, Sleep}
	Memory		{Full, Self-refresh}, Self-refresh mode can be internally invoked from CPU mode. “Full” mode is re-entered when an unmasked interrupt occurs.

The power knobs of the PAMA-PPC architecture and the ranges of each power knob are shown in Table 4. The IBM PowerPC 750FX has two internal PLLs for clock frequency scaling, which allows a clock frequency change to be as fast as in three clock cycles when switching between two stable frequencies from the two internal PLLs.

II.B. Firmware and Software

II.B.1. Network Firmware

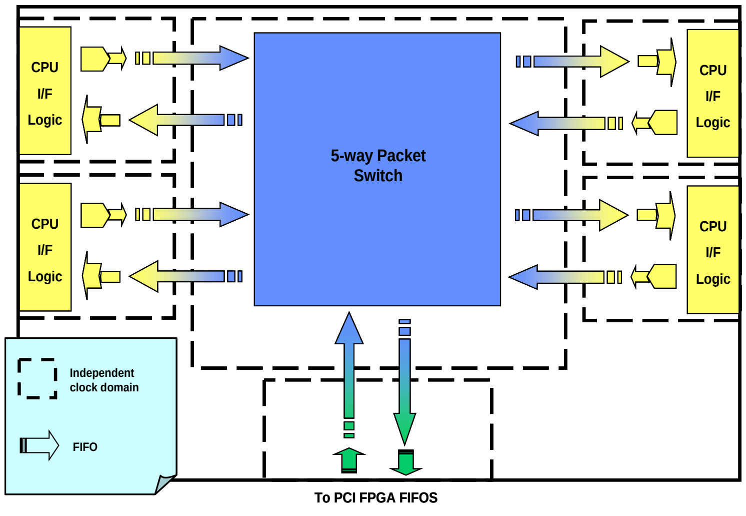


Figure 6. Network firmware architecture

The PAMA network firmware supports communication between the processor nodes. It implements a 5-way crossbar packet switched network. The firmware mimics Ethernet networking and provides an Ethernet physical layer interface to the software running on the processor nodes. The architecture of the network firmware is shown in Figure 6. When we ported the firmware from PAMA-SH to PAMA-PPC, only the processor interface logic had to be changed. The maximum speed of the 5-way packet switch is limited by the speed of the programmable logic we used and the complexity of the interface logic and switch logic. We got a maximum 5-way packet-switch network clock frequency of 33 MHz for the PAMA-SH-4, but 20 MHz for the PAMA-PPC due to the higher complexity of the processor interface logic.

II.B.2. Software

The software architecture for the PAMA architecture is shown in Figure 7. We ported a full-blown Linux for both the SH-4 and the PowerPC 750FX. Linux v.2.4.17 was ported to the SH-4, and Linux v.2.4.22 was ported to the 750FX. MPI (Message Passing Interface)-CH v.1.2.1 was ported to the platform on top of Linux to provide multiprocessor communication. The standard socket interface of Linux is also available for inter-process communications.

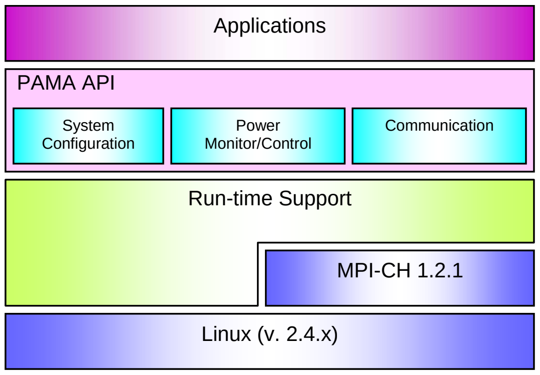


Figure 7. PAMA software architecture

II.B.3. Power Control Library

The PAMA API is an integrated API (Application Programming Interface) for both networking and power management. The PAMA API consists of three components: system configuration, power monitor/control, and communication. The system configuration API includes the PAMA API library initialization and MPI initialization. Power monitor and control functions include voltage scaling, clock frequency scaling and query of remaining energy. MPI-CH provides the actual inter-processor communication. A detailed list of the API is found in the appendix. The PAMA API provides a uniform interface for an application program so that code reusability is greatly improved. Since MPI-CH is a de facto standard for multiprocessor communication, the PAMA API can be easily incorporated into the existing applications using MPI-CH.

II.B.4. Operating System

Linux was chosen for the operating system for PAMA-SH and PAMA-PPC for the following reasons: first, Linux is open source so we can add power-aware features to Linux freely. Second, architecture-specific Linux versions for the Hitachi SH-4 and PowerPC architecture exist. Third, most applications running on Linux and other Unix-like operating systems can run with little porting effort.

Despite the availability of Linux on the SH-4 and PowerPC processors, platform-specific details had to be changed and power-aware features were added. The overall architecture and components of our Linux ports are shown in Figure 8. Architecture specific changes include the interrupt controller, DMA (Direct Memory Access) controller, and console driver. Power management features are implemented as device drivers. The Ethernet interface is used for the inter-processor communication and any applications using a standard Unix network stack without modifications.

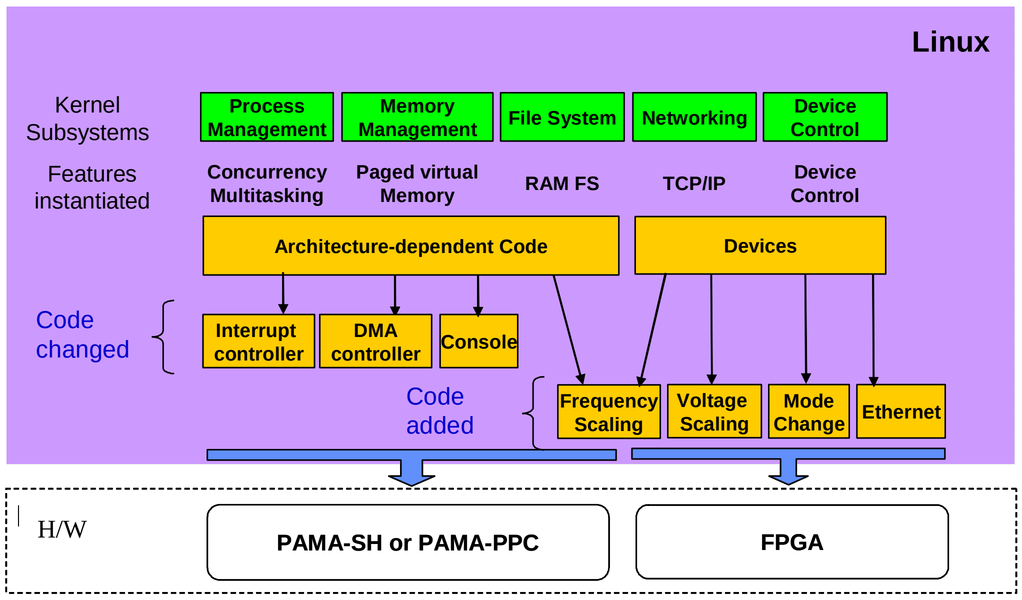


Figure 8. PAMA Linux architecture

III. Power Aware Resource Synthesis Techniques

III.A. Introduction

We developed analytic techniques that synthesize resource mappings, voltages, and number of processors for optimal usage of power and maximum throughput. These techniques can be used to determine power management parameters at compile time for platforms like PAMA. In this section, we first present two analytic techniques for single processor system, and then we present a technique for multi-processor systems. Our technique uses the CMOS (Complimentary Metal Oxide Semiconductor) power-latency model proposed by Chandrakasan et al [3], which is presented below.

$$Latency = C\frac{v}{(v-V_t)^2}, \quad V_t < v \le V_{ref} \qquad \textbf{Eq. 1}$$

$$Energy \approx (\#\text{ of switching}) \times C_{eff} \times v^2 \qquad \textbf{Eq. 2}$$

$$e_i(v_i) = e_{i,ref} \times (\frac{v_i}{V_{i,ref}})^2 \qquad \textbf{Eq. 3}$$

$$l_i(v) = l_{i,ref} \times \frac{v \times (V_{i,ref} - V_{i,t})^2}{V_{i,ref} \times (v - V_{i,t})^2} \qquad \textbf{Eq. 4}$$

In CMOS digital circuits, latency and energy consumption for a task are given by equations (Eq. 1) and (Eq. 2) [3]. In these equations, v denotes supply voltage. V_t and V_{ref} denote threshold voltage and reference voltage, respectively and are inherent to the fabrication technology. C is a technology-dependent constant, and C_{eff} is the effective capacitance.

When latency $l_{i,ref}$ and average energy consumption $e_{i,ref}$ of a task at reference voltage $V_{i,ref}$ of the resource are given, latency $l_i(v_i)$ and average energy consumption $e_i(v_i)$ at the supply voltage v_i can be driven by equations (Eq. 3) and (Eq. 4).

III.B. Single CPU Resource Management Techniques

We developed analytical techniques to synthesize resource utilizations and voltages of tasks in a single processor system. Multiple independent tasks share a single processor system. When a system-wide power/energy budget is limited, running the system within the power/energy budget is the first goal, and achieving the biggest throughput is another goal. We developed a resource usage synthesis technique and a voltage synthesis technique to address those goals.

III.B.1. Resource Usage Synthesis Technique

A set of independent tasks $\{\tau_1, \tau_2, .., \tau_N\}$ exists in a single CPU system. Each task has its own characteristics: frequency of execution (f_i), execution time (t_i), energy consumption per execution instance (e_i), weight (w_i), and performance function (p_i). The performance function of a task is a method of determining a reward function for running the task. We consider non-decreasing concave performance functions. System-wide performance is the sum of the products of the performance function and the weight of each task, $\sum_{i=1}^{N} w_i \times p_i$. The resource utilization ρ_i allocated to task τ_i is the product of its execution time and frequency, $\rho_i = t_i \times f_i$. The energy consumption of the system E is limited by total energy available E_{max} within a time interval T. Our goal is to find an optimal processor utilization allocation such that system performance is maximized and the system is schedulable and the energy consumption is within the energy budget.

We consider two different models for execution time and frequency. One assumes a fixed execution time with a variable frequency, which we call the *Variable Frequency (VF)* model. The other assumes a variable execution time with a fixed frequency, which we call the *Variable Execution Time (VET)* model.

We assume the use of real-time scheduling policies for which schedulability analysis can be done with total resource utilization, such as Earliest Deadline First (EDF), Least Laxity First (LLF), Rate Monotonic (RM), and Rate Monotonic-Harmonic (RM-H) for the VET model. Since the VF model considers the frequency of a task as a free variable, all the real-time scheduling policies usable for the VET model may not be used. An example is the RM-H policy in which the frequencies of the tasks should be harmonic. The maximum resource utilization ρ_{max} depends on the real-time scheduler used.

Under the VF model, each task has a range of operating frequencies [$f_{i,min}$, $f_{i,max}$] but has a fixed execution time t_i. The performance function $p_i(f_i)$ is a function of the frequency f_i. The problem can be stated as follows:

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(f_i)$$

$$\text{subject to } \sum_{i=1}^{N} \rho_i \le \rho_{\max} \text{ and } f_{i,\min} \le f_i \le f_{i,\max} \text{ and } \rho_i = t_i \times f_i \text{ and } E \le E_{\max}.$$

Under the VET model, each task has a range of execution times [$t_{i,min}$, $t_{i,max}$] but has a fixed frequency f_i. The performance function $p_i(t_i)$ is a function of the execution time received t_i. The problem can be stated as follows:

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(t_i)$$

$$\text{subject to } \sum_{i=1}^{N} \rho_i \le \rho_{\max} \text{ and } t_{i,\min} \le t_i \le t_{i,\max} \text{ and } \rho_i = t_i \times f_i \text{ and } E \le E_{\max}.$$

The problems under the two models can be stated in an integrated way in terms of the resource utilization ρ_i of a task τ_i. The performance function of τ_i can be uniformly presented as a function of ρ_i.

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(\rho_i)$$

$$\text{subject to } \sum_{i=1}^{N} \rho_i \le \rho_{\max} \text{ and } \rho_{i,\min} \le \rho_i \le \rho_{i,\max} \text{ and } \rho_i = t_i \times f_i \text{ and } E \le E_{\max}.$$

Now the problem becomes finding resource utilization allocations for the tasks that maximize system performance within the energy budget. The resource utilization of a task is a function of execution time and frequency. Either execution time or frequency can be a free variable depending on the model used.

III.B.2. Motivational Examples

Table 5. Properties of the tasks in a sample control system

	t_i (ms)	$f_{i,min}$ (Hz)	$f_{i,max}$ (Hz)	e_i (mJ)	w_i	$p_i(f_i)$
τ_1	10	10	20	4	1	f_1
τ_2	5	15	50	1	5	$\log(f_2)$
τ_3	7	15	50	10	4	$\sqrt{f_3}$
τ_4	2	20	90	3	25	$1-e^{-2f_4}$
τ_5	9	10	40	1	5	$\sqrt[3]{5f_5}$

Consider a control system that consists of five tasks. The properties of the tasks in the system are summarized in Table 5. For illustrative purposes, we chose a variety of non-decreasing concave functions for the performance functions. The EDF scheduling policy is assumed, and its maximum resource utilization ρ_{max} is 1.0. We assume the energy is refreshed every hour, T = 3600 seconds, and the energy budget in T, E_{max}, is 2000 Joules. Total energy consumption within a time interval T is the sum of energy consumption by the tasks during T, $E = \sum_{i=1}^{N} \lceil T \times f_i \rceil \times e_i$, and it should be within the energy budget E_{max}.

The problem is to find the static frequencies of the tasks such that system performance is maximized under the energy budget:

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(\rho_i)$$

$$\text{subject to } \sum_{i=1}^{N} \rho_i \le 1.0 \text{ and } f_{i,\min} \le f_i \le f_{i,\max} \text{ and } \rho_i = t_i \times f_i \text{ and } E \le E_{\max}.$$

III.B.3. Optimal Throughput-Aware Resource Allocation

In this section, we present an optimal resource utilization allocation technique in terms of resource utilization. The problem is to find resource allocations to maximize throughput per resource usage and it is described as follows:

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(\rho_i) \text{ subject to } \sum_{i=1}^{N} \rho_i \le \rho_{\max} \text{ and } \rho_{i,\min} \le \rho_i \le \rho_{i,\max}$$

We assume the EDF policy for real-time scheduling and the total resource utilization available ρ_{max} is 1.0. Depending on the model used, the following hold:

$\rho_{i,min} = t_{i,min} \times f_i$ and $\rho_{i,max} = t_{i,max} \times f_i$ for the VET model

$\rho_{i,min} = t_i \times f_{i,min}$ and $\rho_{i,max} = t_i \times f_{i,max}$ for the VF model.

Let $\phi_i(\rho_i)$ be the weighted performance of task τ_i at ρ_i, which is $\phi_i(\rho_i) = w_i \times p_i(\rho_i)$. And let $\phi'_i(\rho_i)$ be the first derivative of $\phi_i(\rho_i)$ at ρ_i.

```
Algorithm 1.

(1)  S ← {i| 1 ≤ i ≤ N}
If ($\sum_{i\in S} \rho_{i,\max} \le \rho_{\max}$) [ A Solution is Found! Exit. ]
(3)  If ($\sum_{i\in S} \rho_{i,\min} > \rho_{\max}$) [ No Solution Exists! Exit. ]
(4)  $d \leftarrow \rho_{max}$        /* maximum total resource utilization */
(5)  T ← $\{(i, \phi'_i(\rho_{i,min}))| i \in$ S$\} \cup \{(i, \phi'_i(\rho_{i,max}))| i \in$ S$\}$
(6)  Sort T in descending order of $\phi'_i(\rho)$
(7)  While (S is not empty) [
(8)      If (T is empty) [
(9)              Maximize $\sum_{i\in S} \phi_i(\rho_i)$ subject to $d = \sum_{i\in S} \rho_i$ using Lagrange's method. Exit. ]
(10)     $x \leftarrow$ median $\phi'_i(\rho_i)$ value in T
(11)     For $i \in$ S [
(12)         $\rho^*_i \leftarrow$ max. $(\rho_i, \rho)$ where $\phi'_i(\rho) = x$
(13)         if ($\rho^*_i \ge \rho_{i,max}$) [ $\rho^*_i \leftarrow \rho_{i,max}$ ] ] /* End of For */
(14)     If ($d \ge \sum_{i\in S} \rho_i^*$) [
(15)         For ($i \in$ S) [ $\rho_i \leftarrow \rho^*_i$ ]
(16)         If ($d = \sum_{i\in S} \rho_i$) [ The Solution is found!! Exit. ]
(17)         $Low \leftarrow x$ ; Q ← $\{i | \rho_i = \rho_{i,max}\}$ ; P ← $\{(i, \phi'_i(\rho_i))| \phi'_i(\rho_i) \le x\}$ ] /* End of If */
(18)     Else ($d < \sum_{i\in S} \rho_i^*$)[
(19)         $High \leftarrow x$ ; Q ← $\{i | \rho^*_i = \rho_{i,min}\}$ ; P ← $\{(i, \phi'_i(\rho_i))| \phi'_i(\rho_i) \ge x\}$ ] /* End of Else */
(20)     S ← S – Q;      $d \leftarrow d - \sum_{i\in Q} \rho_i$
(21)     T ← T– ($\{(i, \phi'_i(\rho_i))| i \in$ Q and $\rho_i = \rho_{i,min}\} \cup \{(i, \phi'_i(\rho_i))| i \in$ Q and $(\rho_i = \rho_{i,max})\}$)
(22)     T ← T – P ] /* End of While */
```

The main idea of the algorithm is to find an optimal value of $\phi'_i(\rho_i)$ using binary search in the search space of $\min(\phi'_i(\rho_{i,min}))$ and $\max(\phi'_i(\rho_{i,max}))$ where $1 \le i \le N$. Initially the algorithm checks the trivial cases where maximum possible resource utilization is smaller than system utilization allowed as shown in line (2) or where minimum resource utilization required is larger than system resource utilization allowed as shown in line (3). The *While* loop in lines (7) through (22) is a binary search algorithm which reduces the search space by half in each iteration. The size of the search space is at most $2N$ where N is the number of tasks in the system. In each iteration of the *While* loop, the median $\phi'_i(\rho_i)$ value in set **T** is chosen and stored in x in line (10). If the maximum $\phi'_i(\rho_i)$ value of a task is smaller than x, its resource utilizations are set at their maximum value as is shown in lines (12) and (13). When the sum of the resource utilizations at x is smaller than the maximum total resource utilization, the optimal value of $\phi'_i(\rho_i)$, $i \in \mathbf{S}$, is higher than x, so x is set to the lower bound of the search space. The tasks with their resource utilizations equal to their maximum value are removed from further consideration, which are shown in lines (14)–(17) and (20)–(22). When the energy consumption is equal to the energy constraint, the optimal solution is found shown in line (16). If the sum of the resource utilization at x is larger than the maximum total resource utilizations, x is set to the upper bound, and the tasks with current resource utilizations equal to their minimum value are removed from further consideration as shown in lines (18)–(19) and (20)–(22). When set **T** becomes empty as shown in line (8), the upper bound '*High*' and the lower bound '*Low*' of $\phi'(\rho)$ value are given. Lagrange's method is used to find an optimal solution as shown in line (9).

The number of iterations of the *While* loop in line (7) is O(log N) because it is a binary search algorithm. Inside the *While* loop, the running time of all lines except line (9) and the *For* loop in (11) is at most O(N). When we assume the existence of the inverse function of $\phi'_i(v_i)$, the running time of the *For* loop in line (11) becomes O(N). The unique Lagrange multiplier can be solved in linear time under the concavity assumption [4]. Therefore, the time complexity of our algorithm is O(N log N), which is an improvement over the O(N^2 log N) in [5]. The optimality of this algorithm is proven in [6].

III.B.4. Optimal Power-Aware Resource Allocation

In this section, we present an optimal resource utilization allocation technique in terms of energy usage. The problem is to find resource allocations to maximize throughput per energy consumption under resource and energy constraints and is described as follows:
In this section, we present a fast resource utilization allocation technique that solves the following problem:

$$\text{Maximize } \sum_{i=1}^{N} w_i \times p_i(\rho_i) \text{ subject to } \sum_{i=1}^{N} \rho_i \le \rho_{\max} \text{ and } \sum_{i=1}^{N} E_i \le E_{\max} \text{ and } \rho_{i,\min} \le \rho_i \le \rho_{i,\max}.$$

The energy consumption E_i of a task τ_i within an interval T is the product of energy consumption per execution instance and the maximum number of execution instances within the interval:

$$E_i = \lceil T \times f_i \rceil \times e_i.$$

The ceiling function is used to consider the worst case when the interval is not divisible by the period of a task; however, we approximate the energy consumption without a ceiling function. This is acceptable when the interval is much larger than the period of a task, which is often the case in control systems. The energy consumption of a task under the VF model is a function of the resource utilization allocation ρ_i and is also presented as $E_i(\rho_i)$.

$$E_i(\rho_i) = \lceil T \times f_i \rceil \times e_i = \left\lceil \frac{T \times \rho_i}{t_i} \right\rceil \times e_i \cong \frac{T \times \rho_i \times e_i}{t_i}$$

Under the VET model, energy consumption changes as the execution time changes. We assume that the energy consumption of a task is linearly proportional to its execution time. Tasks may have different energy consumption densities per unit time. Then energy consumption of a task under the VET model is a function of the resource utilization allocation ρ_i as shown below where $e_{i,min}$ denotes the energy consumption at the minimum execution time of task τ_i.

$$E_i(\rho_i) = \lceil T \times f_i \rceil \times e_i \cong T \times f_i \times e_i = T \times f_i \times \rho_i \times \frac{e_{i,\min}}{\rho_{i,\min}}$$

The energy consumption $E_i(\rho_i)$ of a task τ_i is a function of ρ_i for both the VF and the VET models. We assume the use of an ideal on-off energy saving technique. When the system is idle, we assume that energy is not wasted at all. Our technique derives a static execution time or a frequency of a task for higher performance within the energy budget. Our technique is different from power minimization synthesis techniques, because our goal is not energy reduction but maximum performance.

Algorithm 1 is an optimizing algorithm in terms of resource utilization. Algorithm 1 can be modified to be an optimizing algorithm in terms of energy consumption by changing the definition of $\phi'_i(\rho)$

from $\frac{d\phi_i}{d\rho_i}$ to $\frac{d\phi_i}{dE_i}$. We call the algorithm Algorithm 2 and rewrite the lines that need to be changed from Algorithm 1.

```
Algorithm 2.
(2)  If ($\sum_{i\in S} E_i(\rho_{i,\max}) \le E_{\max}$) [ the Solution is Found! Exit. ]
(3)  If ($\sum_{i\in S} E_i(\rho_{i,\min}) > E_{\max}$) [ then No Solution Exists! Exit. ]
(4)  $d \leftarrow E_{max}$
(9)            Maximize $\sum_{i\in S}\phi_i(\rho_i)$ subject to $d = \sum_{i\in S} E_i(\rho_i)$ using Lagrange's method. Exit. ]
(14)     If ($d \ge \sum_{i\in S} E_i(\rho^*_i)$)
(16)        If ($d = \sum_{i\in S} E_i(\rho_i)$) [ the Solution is found!! Exit. ]
(18)     Else ($d < \sum_{i\in S} E_i(\rho^*_i)$)
(20)     S ← S – Q;  $d \leftarrow d - \sum_{i\in Q} E_i(\rho_i)$
```

III.B.5. Near Optimal (Power and Throughput)-Aware Resource Allocation

Both Algorithm 1 and Algorithm 2 are still incomplete in the sense that they can produce infeasible solutions. Algorithm 1 can produce a solution consuming more energy than the energy constraint E_{max}. Likewise, Algorithm 2 can produce a solution for which resource usage is over the limit ρ_{max}. We can easily change both algorithms to produce a feasible solution by adding conditions that check if both resource usage and energy consumption are under the limits. Then, the algorithms end when either resource usage reaches its limit or energy consumption reaches the total energy budget. Let the revised algorithms be Algorithm 1* and Algorithm 2*.

The solutions produced by the revised algorithms may not be optimal. We can draw the solution space as a two dimensional graph as shown in Figure 9 (a) where the x-axis corresponds to the energy consumption of a solution and the y-axis corresponds to the resource usage of a solution. The point S in Figure 9 (a) denotes the starting point where all the tasks are given their minimum resource shares. A solution produced by Algorithm 1* may not be optimal when the total resource usage of the solution is less than ρ_{max} and the energy consumption is equal to E_{max}, which is shown as the point A^* in Figure 9 (a). Likewise, a solution produced by Algorithm 2* may not be optimal if the energy consumption of the solution is less than E_{max} and the total resource usage is equal to ρ_{max}, which is shown as the point B^* in Figure 9 (a). An optimal solution exists on the lines ($x = E_{max}$, $\rho_A \le y \le \rho_{max}$) and ($E_B \le x \le E_{max}$, $y = \rho_{max}$) shown in bold in Figure 9 (a).

We may improve the quality of the solution using those algorithms with one more step, which is shown in Algorithm 3. The basic idea is shown in Figure 9 (c). The behaviors of Algorithm 1* and Algorithm 1 are depicted in Figure 9 (a) and Figure 9 (b) respectively. While Algorithm 1* stops at the point A^*, Algorithm 1 stops at A shown in Figure 9 (b). Likewise, Algorithm 2 stops at the point B shown in Figure 9 (b). Both solutions at A and B are infeasible. Algorithm 3 improves the solution from the point A to the point A^{**} by decreasing the resource shares of the tasks in a way that is similar to Algorithm 2. The resource shares of the tasks having the smallest $\phi_i'(\rho_i) = \frac{d\phi_i}{dE_i}$ value are decreased from the point A to A^{**}. In other words, it chooses the tasks with the minimum performance reduction at the expense of energy reduction. Likewise, it finds the point B^{**} from B using an algorithm similar to but opposite in direction of Algorithm 1.

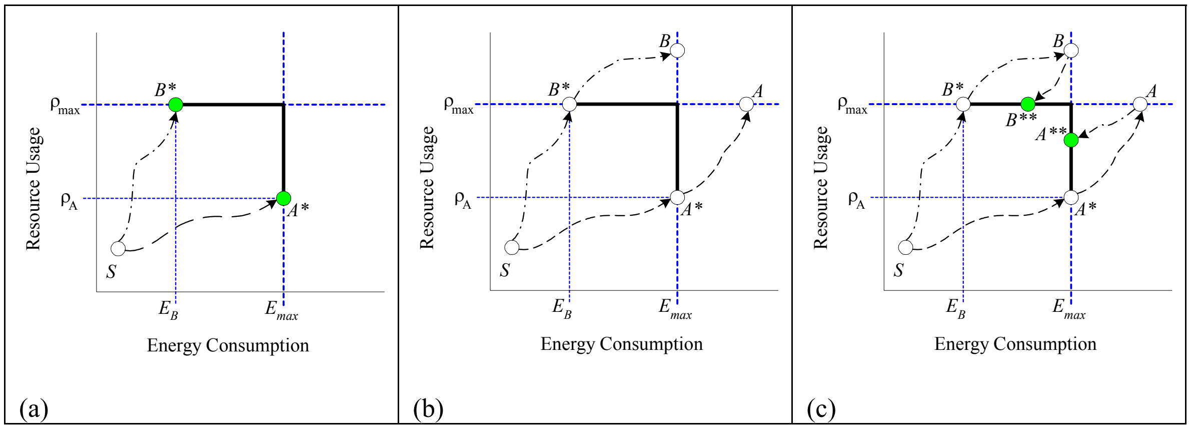


Figure 9. Exploration of the solution space

```
Algorithm 3:
(1) If ($\sum_{i=1}^{n} E_i(\rho_{i,\max}) \le E_{\max}$ and $\sum_{i=1}^{N} \rho_{i,\max} \le \rho_{\max}$), then it is optimal at $\rho_{i,max}$ for all tasks. Exit.
(2) Load-based search:
(2-1)     Run Algorithm 1 and let the resulting resource share of task $\tau_i$ be $\rho_{i,sl}$.
If ($\sum_{i=1}^{n} E_i(\rho_{i,sl}) \le E_{\max}$), then it is optimal.  Exit.
Run Algorithm 2 with T ← {(i, $\phi'_i(\rho_{i,sl})$)} in line (5) /* Reduce resource share until $\sum_{i=1}^{N} \rho_i = \rho_{\max}$ */
 (3) Energy-based search:
(3-1)     Run Algorithm 2 and let the resulting resource share of task $\tau_i$ be $\rho_{i,se}$
    (3-2)     If ($\sum_{i=1}^{N} \rho_{i,se} \le \rho_{\max}$) then it is optimal. Exit.
Run Algorithm 1 with T ← {(i, $\phi'_i(\rho_{i,se})$)} in line (5) /* Reduce energy usage until $\sum_{i=1}^{n} E_i(\rho_i) = E_{\max}$ */

(4) Choose the solution with higher performance between the two solutions from lines (2-3) and (3-3).
```

III.B.6. Motivational Example Results

Table 6. Solutions of the motivational example

Algorithm 1*			Algorithm 2*			Algorithm 3		
	ρ_i	f_i (Hz)		ρ_i	f_i (Hz)		ρ_i	f_i (Hz)
τ_1	0.200	20.0	τ_1	0.200	20.0	τ_1	0.200	20.0
τ_2	0.107	21.5	τ_2	0.25	50.0	τ_2	0.184	36.3
τ_3	0.263	37.6	τ_3	0.150	21.4	τ_3	0.227	32.6
τ_4	0.040	20.0	τ_4	0.040	20.0	τ_4	0.040	20.0
τ_5	0.160	17.7	τ_5	0.360	40.0	τ_5	0.351	39.0

The solutions of the motivational example in the start of Section III.A.1 are shown in Table 6 and Table 7. The solution derived by Algorithm 2* has higher performance than the solution by Algorithm 1*, which means that energy-based optimization works better than load-based optimization for the example. Algorithm 3 improves the solutions derived by Algorithm 1 and Algorithm 2. The solution derived by Algorithm 1 is improved and its performance becomes 112.1. The solution derived by Algorithm 2 is improved and its performance is 114.8. Algorithm 3 chose the solution with higher performance, which is shown in Table 6 and Table 7.

Table 7. Resource utilization, energy consumption, and performance of the solutions

	Algorithm 1*	Algorithm 2*	Algorithm 3
Total resource utilization	0.770	1.000	1.000
Total energy consumption (Joule)	2000	1600	1950
Total Performance	107.2	112.3	114.8

III.C. Voltage Synthesis Technique

Voltage scaling is a major source of power management. We propose an optimal voltage synthesis technique for single processor systems. Our technique synthesizes optimal voltages at a given time which results in maximum system throughput within energy constraints. We chose a LEOS (Low Earth Orbit Satellite) application to demonstrate our technique. The path of a LEOS is pre-defined and periodic. The tasks to be performed in the next period are stochastically predictable, and energy budget in a period is strictly limited but predictable.

III.C.1. Motivational Example

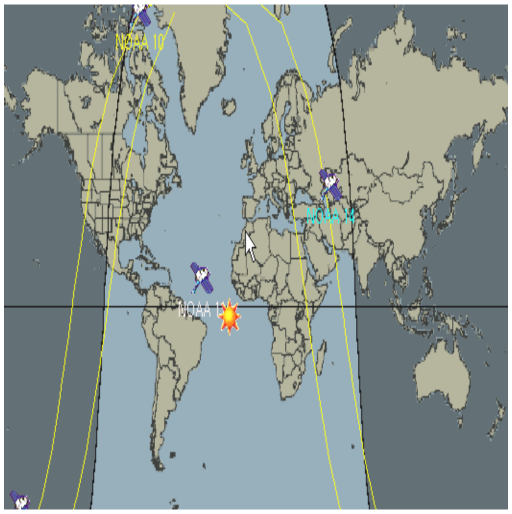


Figure 10. Screenshot of Jtrack

Table 8. Characteristics of the system per region

Region	R_1	R_2	R_3	R_4	R_5	R_6	R_7
d_i (min.)	26.1	11.6	11.6	8.7	14.5	11.6	20.3
λ_i	0.4	1	0.2	0.4	1.2	0.4	0.2
$\mu_{i,ref}$	1	1.5	1	1	1.8	1	1
w_i	1	1.5	0.9	0.7	1.2	0.7	0.7
$e_{i,ref}$ (mJ)	2.1	2.5	1.8	2.1	2.5	2.1	1.8
B	100	50	80	100	70	100	80
Λ_i	0.1	0.02	0.1	0.05	0.01	0.05	0.1

Table 9. System constraints

V_{ref}	V_t	E_{Total}
3.3V	0.6V	3300 J

As an example of a LEO satellite, the weather satellite NOAA10 tracks the following regions on January 26th, 2001 at around 11:00am. The image in Figure 10 is from NASA's J-Track application [7]. The satellite covered seven different regions during the period.   Region R_1 corresponds to the Pacific Ocean, R_2 to North America, R_3 to the Arctic and Greenland, R_4 to the Atlantic Ocean, R_5 to Europe and Africa, R_6 to the Atlantic Ocean, and R_7 to the Antarctic. Based on the characteristics of similar satellite NOAA12, its period is 101.3 minutes and it's in polar orbit.

The synthetic characteristics in each region are shown in Figure 10. The duration d_i at region R_i is the length of the satellite's stay over the region. The arrival rate λ_i at region R_i is the average arrival rate of inputs per millisecond. The service rate $\mu_{i,ref}$ at region R_i is the average number of inputs that can be processed per millisecond at the reference voltage V_{ref} shown in Table 8. The weight w_i of an input for region R_i denotes how valuable an output is when compared to the outputs in other regions. Energy consumption $e_{i,ref}$ for region R_i is the average energy at the reference voltage V_{ref} consumed to process an input in milli-Joules. Energy consumption is assumed to depend on the service time. The buffer size B denotes how many inputs can be stored in the buffer that may vary at different regions. The maximum input loss probability Λ_i for region R_i is the maximum allowed input loss probability for the region.

As an example, the second column labeled R_1 in Table 8 is interpreted as follows. The satellite flies over the Pacific Ocean for 26.1 minutes. During that time 0.4 inputs arrive per millisecond on the average. The satellite can process one input per millisecond on the average in the region at the reference voltage. The average energy consumption for processing one input is 2.1 milli-Joules in the region. The system can hold up to 100 inputs in its buffer. Any more inputs are discarded. Due to the limited buffer space, input loss is expected. However, there is maximum allowable input loss probability for each region. Up to 10 percent of inputs gathered during the flight over Pacific Ocean can be discarded.

System-wide constraints are described in Table 9. According to Table 9, the satellite can use up to its maximum energy budget E_{Total}, which is 3300 Joules, for computation within a period. The system is assumed to be voltage schedulable between its reference voltage V_{ref} , 3.3 Volts, and its threshold voltage V_t, 0.6 Volt, shown in Table 9. Our interest lies in how to derive the optimal voltage for each region that produces maximum performance with a given energy constraint E_{Total}. Performance is defined as the sum of inputs processed multiplied by their weights. The solution to the example is derived using our optimal algorithm of Section III.C.2.

III.C.2. System Constraints and Problem Overview

System constraints consist of performance constraints and energy constraints. Performance constraints are given for each region R_i and they are described as (1) maximum input loss probability Λ_i, shown below and (2) the sum of the products of weight and the number of outputs per region shown below. $P_{i,loss}$ denotes the probability of input loss in the region R_i due to the limited buffer space. The weight of an output w_i in region R_i denotes the importance of an output compared with the outputs of other regions. T_i denotes the duration of region R_i. The energy constraint E_{Total} is the total amount of energy that can be used for the processing in a period shown in below, where E_i denotes energy consumption in region R_i. The design goal is to find feasible voltages for the regions which maximize performance while the energy consumption is within the energy budget E_{Total} by tuning voltages and corresponding clock frequencies in the regions.

$$\forall R_i, \quad p_{i,loss} \le \Lambda_i$$

$$Performance = \sum_i \Omega_i = \sum_i T_i \times w_i \times \lambda_i \times (1 - P_{i,loss})$$

$$\sum_i E_i = \sum_i T_i \times \lambda_i \times (1 - p_{i,loss}) \times e_i \le E_{Total}$$

We propose an algorithm to solve the problem, which is shown below. The algorithm first finds an initial solution that satisfies input loss constraints of all regions. Each region is evaluated independently and has a single solution. If the resulting solution does not satisfy energy constraints, there is no feasible solution, which is shown in lines (1) through (8). The while loop in lines (16) through (40) is a binary search algorithm which reduces the search space by half as shown in line (20). The size of the search space is at most 2N where N is the number of regions in the period. At each iteration of the while loop, the median $\phi_i(v)$ value Φ in set T is chosen and the corresponding voltage is estimated in lines (20) through (24). If the $\phi_i(v)$ value of a region at the reference voltage is smaller than Φ, its voltage is set at the reference voltage. When the energy consumption at Φ is smaller than the energy constraint E_{Total}, Φ is set to the lower bound of the search space and the regions with their current voltages equal to the reference voltage is removed from further consideration, which is shown in lines (30)–(32) and (37)–(40). When the energy consumption is equal to energy constraint, the optimal solution is found shown in line (29).

Otherwise, Φ is set to the upper bound, and the regions with current voltages equal to the initial voltages are removed from further consideration as shown in lines (34)–(36) and (37)–(40). When set T becomes empty as shown in line (18), the upper bound 'High' and the lower bound 'Low' of $\phi(v)$ are given. Lagrange's method to find an optimal solution is used to find an optimal solution as shown in line (19). The optimality of the algorithm is proved in [8].

We present a simple example to show the behavior of the algorithm. Assume that there are two regions in a period R_1, R_2, $\phi_1(V_{1,bot}) = a_1$, $\phi_1(V_{1,ref}) = b_1$, $\phi_2(V_{1,bot}) = a_2$, $\phi_2(V_{2,ref}) = b_2$, $a_1 < a_2 < b_1 < b_2$, and their $\phi_i(v)$ values are drawn in Figure 11. In its first iteration of the while loop shown in Figure 11 (a), there are four unique ϕ values in **T** and the third largest ϕ value in **T** is chosen as Φ that is a_2. Energy consumption at Φ is smaller than the energy constraint. According to lines (31) and (32), **Q** = {} and **P** = {(1, a_1),(2, a_2)}, and **T** becomes {(1, b_1), (2, b_2)}. In the second iteration, the algorithm probes b_1 as shown in Figure 11 (b). Now, the energy consumption at b_1 is larger than the energy constraint. According to lines

(35) and (36), **Q** = {} and **P** = {(1, b_1), (2, b_2)}. Since set **T** is empty in Figure 11 (c), the algorithm finds optimal voltages $v_{1,opt}$ and $v_{2,opt}$ using Lagrange's method, which is shown in line (19).

Algorithm 4. Optimal Voltage Assignment for regions in a period

```
(1)  S ← {$i$| $R_i$ }; $E$ ← $E_{Total}$
(2)  For $i \in$ S
(3)      Calculate $v_i$, $\phi_i(v_i)$ and $E_i(v_i)$ where $p_{i,loss} = \Lambda_i$
(4)      $V_{i,bot}$ ← $v_i$
(5)      If ($v_t > V_{i,ref}$)
(6)          No feasible solution exists!! Exit.
     End for
(7)  If ($E < \sum_{i \in S} E_i(v_i)$)
(8)      No feasible solution exists!! Exit.
(9)  If ($E \geq \sum_{i \in S} E_i(V_{i,ref})$)
(10)     Solution found!! Exit.
(11) Q ← {$i$ | $v_i = V_{i,ref}$}
(12) S ← S – Q
(13) $E \leftarrow E - \sum_{i \in Q} E_i$
(14) T ← {($i$, $\phi_i(v)$)| $i \in$ S and ($v = V_{i,ref}$ or $v = V_{i,bot}$)}
(15) Sort T in descending order of $\phi_i(v)$
(16) While (S is not empty)
(17)     $m$ ← number of unique $\phi_i(v)$ values in T
(18)     if ($m$ = 0)
(19)             Maximize $\sum_{i \in S} \Omega_i$
                 subject to $E = \sum_{i \in S} E_i(v_i)$
                 using the method of Lagrange multipliers
         End if
(20)     $\Phi$ ← ($\lfloor m/2 \rfloor$+1)-th largest $\phi_i(v)$ value in T
(21)     For $i \in$ S
(22)         $v'_i$ ← $v$, where $\phi_i(v'_i) = \Phi$
(23)         if ($v'_i > v_{i,ref}$)
(24)             $v'_i$ ← $v_{i,ref}$
(25)     End for
(26)     If ($\Psi \geq \sum_{i \in S} E_i(v'_i)$) /* Upper Half */
(27)         For ($i_i \in$ S)
(28)             $v_i$ ← $v'_i$
(29)         If ($E = \sum_{i \in S} E_i(v'_i)$) Solution found!! Exit
(30)         $Low$ ← $\Phi$
(31)         Q ← {$i$ | $v_i = V_{i,ref}$}
(32)         P ← {($i$, $\phi_i(v)$)| $\phi_i(v) \leq \Phi$}
(33)     Else ($E < \sum_{i \in S} E_i(v'_i)$) /* Lower Half */
(34)         $High$ ← $\Phi$
(35)         Q ← {$i$ | $v'_i = V_{i,bot}$}
(36)         P ← {($i$, $\phi_i(v)$)| $\phi_i(v) \geq \Phi$}
         End if
(37)     S ← S – Q
(38)     $E \leftarrow E - \sum_{i \in Q} E_i(v_i)$
(39)     T← T– {($i$, $\phi_i(v)$)| $i \in$ Q and ($v = V_{i,bot}$ or $V_{i,ref}$)}
(40)     T ← T - P
     End while
```

III.C.3. Solution of the Motivational Example

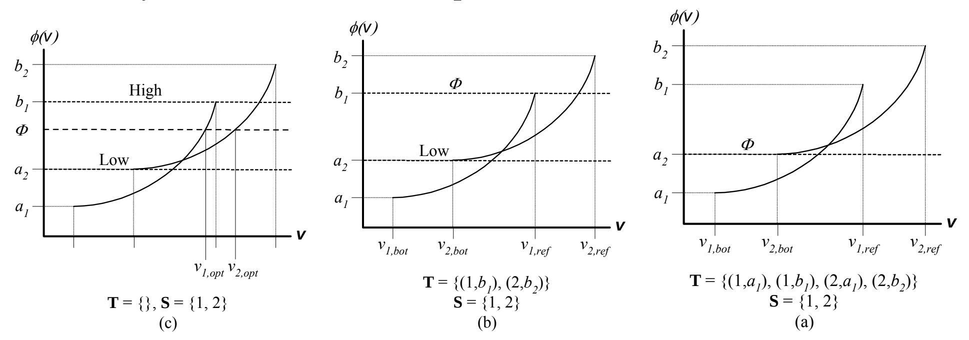


Figure 11. Simple example using Algorithm 4

We used Algorithm 4 to solve the motivational problem shown in Table 8 and Table 9. We used a binary search algorithm to find Lagrange's multiplier in line (19) of Algorithm 4 with 0.0001 percent error bound. It takes less than 8 milliseconds to find the solution on a Pentium III 733-MHz PC running Linux.

Table 10. Solution to motivational example

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
V_d	1.795	2.528	1.374	1.844	2.544	1.844	1.340
Λ_i	0.1	0.02	0.1	0.05	0.01	0.05	0.1
$p_{i,loss}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075

III.C.4. Simulation

The analytical framework assumes there is no interference between two neighboring regions. However, this may not be true. Region R_i can interfere with R_{i+1} when the queue is not empty or the processor is not idle at the time of the region change. In this situation, the processor is working on data collected in region R_i even though the satellite is flying over region R_{i+1}. This situation gets worse when region R_i is highly loaded and the duration of region R_{i+1} is short.

To observe the interference between regions, we simulated the behavior of the motivational example and its variants. Three queue management policies at the boundaries of the regions are simulated which include (1) no management, (2) drop all inputs in the queue except the one that is being processed, and (3) drop all inputs including the one that is currently processed. The simulated result of the example is shown in Table 11 In Table 11(a), $p_{i,loss}$ denotes the loss probability given by the analysis and $p_{i,loss(i)}$ denotes the average packet loss probability observed by the simulation using the *i-th* queue management policy at the boundaries of the regions. Likewise, $\sigma_{i,loss\ (1)}$ denotes standard deviation of the packet loss probability using the *i-th* queue management policy.

Our analytic result does not depend on how large the arrival rate and service rate are but does depend on their ratio or load. To observe the influence of a large arrival rate and service rate and the opposite, we scaled service rate and arrival rate by 10, 0.1 and 0.01 while their ratio remains the same. Their results are summarized in Table 11, Table 12, and Table 13. From those tables, we observe that our analysis technique predicts well for high arrival and service rates. At low arrival and service rates, our analytical result deviates from simulation results. As the arrival and service rates decrease as shown in Table 13, interdependency among the regions at the boundaries between regions gets bigger and the better results are predicted for queue management policies (2) and (3) than no management policy (1).

Table 11. Simulation result of the example.

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$p_{i,loss}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (1)}$	0.10	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (2)}$	0.10	0.02	0.02	0.05	0.01	0.05	0.076
$p_{i,loss\ (3)}$	0.10	0.02	0.02	0.05	0.01	0.05	0.076
$\sigma_{i,loss\ (1)}$	2e-3	1e-3	3e-3	3e-3	7e-3	3e-3	3e-3
$\sigma_{i,loss\ (2)}$	2e-3	1e-3	3e-3	3e-3	7e-4	3e-3	3e-4
$\sigma_{i,loss\ (3)}$	2e-3	1e-3	3e-3	3e-3	7e-4	3e-3	3e-4

(a)

	Energy	$\sigma_{energy,}$	Performance	$\sigma_{performance}$
(1)	3.3e6	1748	3.4e6	1750
(2)	3.3e6	1754	3.4e6	1764
(3)	3.3e6	1752	3.4e6	1763

(b)

Table 12. Simulation result. (10 times service rate and arrival rate)

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$p_{i,loss}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (1)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (2)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (3)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$\sigma_{i,loss\ (1)}$	6e-4	3e-4	8e-4	9e-4	2e-4	9e-4	7e-4
$\sigma_{i,loss\ (2)}$	6e-4	3e-4	8e-4	9e-4	2e-4	9e-4	7e-4
$\sigma_{i,loss\ (3)}$	6e-4	3e-4	8e-4	9e-4	2e-4	9e-4	7e-4

(a)

	Energy	$\sigma_{energy,}$	Performance	$\sigma_{performance}$
(1)	33.0e6	5627	34.3e6	5404
(2)	33.0e6	5785	34.3e6	5666
(3)	33.0e6	5785	34.3e6	5666

(b)

Table 13. Simulation result. (1/10 times service rate and arrival rate)

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$p_{i,loss}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (1)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.074
$p_{i,loss\ (2)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.076
$p_{i,loss\ (3)}$	0.1	0.02	0.02	0.05	0.01	0.05	0.076
$\sigma_{i,loss\ (1)}$	5e-3	3e-3	8e-3	9e-3	2e-3	8e-3	8e-3
$\sigma_{i,loss\ (2)}$	5e-3	3e-3	9e-3	9e-3	2e-3	8e-3	8e-3
$\sigma_{i,loss\ (3)}$	5e-3	3e-3	9e-3	9e-3	2e-3	8e-3	8e-3

(a)

	Energy	$\sigma_{energy,}$	Performance	$\sigma_{performance}$
(1)	0.33e6	552	0.34e6	562
(2)	0.33e6	564	0.34e6	564
(3)	0.33e6	565	0.34e6	563

(b)

Table 14. Simulation result. (1/100 times service rate and arrival rate)

	R_1	R_2	R_3	R_4	R_5	R_6	R_7
$p_{i,loss}$	0.1	0.02	0.02	0.05	0.01	0.05	0.075
$p_{i,loss\ (1)}$	0.11	0.05	8e-3	0.05	0.03	0.03	0.064
$p_{i,loss\ (2)}$	0.10	0.02	0.04	0.06	0.01	0.06	0.080
$p_{i,loss\ (3)}$	0.10	0.02	0.04	0.06	0.01	0.06	0.081
$\sigma_{I,loss\ (1)}$	2e-3	1e-2	2e-2	3e-2	7e-3	2e-2	0.026
$\sigma_{I,loss\ (2)}$	2e-3	1e-2	1e-2	3e-2	3e-2	2e-2	0.026
$\sigma_{I,loss\ (3)}$	2e-3	1e-2	2e-2	3e-2	7e-3	2e-2	0.026

(a)

	Energy	$\sigma_{energy,}$	Performance	$\sigma_{performance}$
(1)	32.4e3	198	33.7e3	201
(2)	32.9e3	175	34.2e3	177
(3)	32.9e3	175	32.9e3	177

(b)

III.D. Dynamic Power Management of Multiprocessor Systems

In this section, we consider a multiprocessor system that has a rechargeable battery to store energy and external power sources. In this system, not only low-power design is important, but also high power utilization and performance are important to achieve. The power management algorithm needs to find an initial power management schedule, and, during run time, it needs to determine system parameters and update the schedule to accommodate the variances of the planned schedule and real schedule. Figure 12 shows our approach. It first estimates the initial power allocation. The estimation of the initial power allocation maximizes power utilization and performance by avoiding oversupplied or undersupplied power conditions by using or saving energy before such conditions occur. Then, based on the power allocation, the system parameters are computed. The parameters are computed to maximize the performance for a given power allocation. Then, during run-time, the variances of the externally supplied energy and power usage are computed dynamically and are used to recalculate the power allowance and system parameters.

III.D.1. Problem Definition

The system considered in this paper consists of *N* homogeneous processors. Each processor can operate at a frequency between f_{min} and f_{max} and can be turned off independently. The supplied voltage to each processor is between v_{min} and v_{max}. The power is supplied from a rechargeable battery that is charged by an external power source that has a periodic power supply schedule. For example, a signal processing system on a satellite operates using energy from a solar panel mounted on the satellite. The charging property is periodic due to periodic orbiting.

Let us denote the period as *T*. For a given time *t*, $0 \leq t < T$, the following are defined:

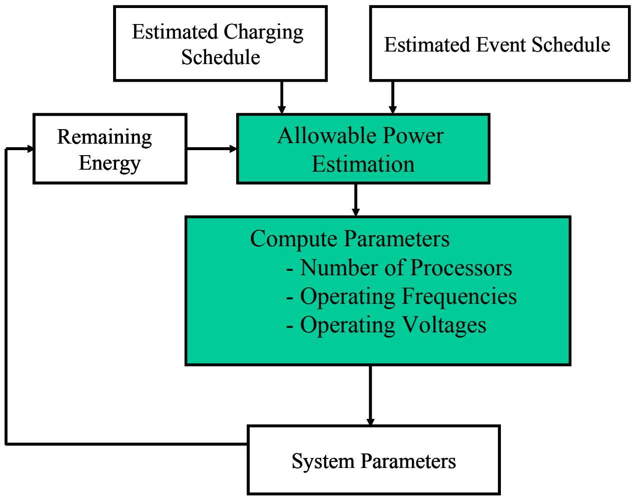


Figure 12. Dynamic power management algorithm

Expected charging schedule *c*(*t*). The *c*(*t*) is an estimated external power that will be supplied at time *t*. The schedule may be derived theoretically or empirically. For example, the recorded charging power for the previous period or weighted average of the several previous periods can be used.

Expected event rate schedule *u*(*t*): The event rate is the rate of the events that initiate the computation on the system. As in the expected charging schedule, *u*(*t*) can be derived using any reasonable method such as predicting from data for the previous periods and mathematically computed supply power. For example, if the event is related to weather conditions, weather forecast data can be used for the estimation of *u*(*t*).

Weight function *w*(*t*): The weight function is user input that is used to emphasize some portion of the period. For example, if we want to process data more intensively during commute time in a traffic monitoring system, then the period is given a higher weight value.

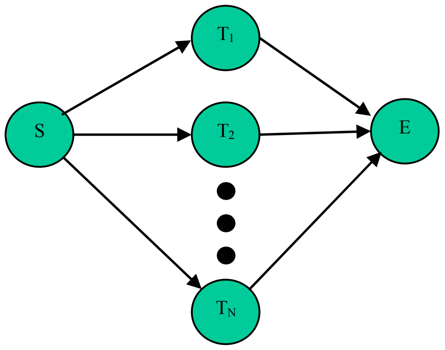


Figure 13. Task graph

Maximum charging capacity c_{max}: The maximum capacity of the rechargeable battery. Even if the externally supplied energy is available for charging, if the energy charged to the battery is equal to c_{max}, then the energy cannot be stored. Thus, the additional supplied energy is wasted.

Minimum charging capacity c_{min}: The minimum charge that should be maintained at all times.

Energy utilization: (*Energy used for computation*)/(*Energy available*) for the period T

The goal is to compute the following parameters of the multiprocessor system to maximize the energy utilization and maximize the performance for the given inputs.

- Number of processors n_t: The number of active processors on the system at time t.
- Frequncy of processors $(f_0, f_1, \ldots f_{n-1})_t$: The frequency of each active processor at time t.
- Voltage of processors $(v_0, v_1, \ldots v_{n-1})_t$: The voltage of each active processor at time t.

III.D.2. Performance and Power Model

The performance of the system as a function of frequency f and voltage v is modeled as follows[1]:

$Perf(f,v) \propto \min(f, g(v))$, $f_{\min} \le f \le f_{\max}, v_{\min} \le v \le v_{\max}$, where $g(v)$ is the maximum frequency that can be obtained when voltage v is applied. The equation indicates that the performance is proportional to the frequency if f is less than $g(v)$. The applications that we considered are parallel applications with initial and final stages. The task graph is shown in Figure 13. The performance of the system for the applications using n homogeneous processors is derived from Amdahl's law [9]:

$$Perf(n) = c_0 \frac{1}{T_s + \frac{T_t - T_s}{n}}$$

where c_0 is a constant, T_t is a total execution time when n = 1, and T_s is the execution time of the portion that cannot be parallelized. When we consider both the frequency and voltage, the overall performance equation is

$$Perf(n,f) = c_1 \frac{\min(f, g(v))}{T_s + \frac{T_t - T_s}{n}}$$

The power consumption model for a uniprocessor [10] is

$$Power(n,f,v) \propto f_i {v_i}^2$$

Thus, the power consumption for N processors is

$$Power(n,f,v) = c_2 \sum_{i=0}^{n-1} f_i {v_i}^2$$

where c_2 is a constant[2]. For a homogeneous system in which all the processors operate at the same clock frequency and voltage, then the power consumption is

$$Power(n,f,v) = c_2 nfv^2$$

This equation indicates the power is proportional to the number of processors, operating frequency, and square of operating voltage.

III.D.3. Dynamic Power Management Technique

In this section, our dynamic power management technique is described. First, the initial power allocation computation is presented. Then, the system parameter computation algorithm based on the

[1] Note that the equation is simplified for analysis by assuming other factors such as memory latency and data dependency are all scalable to the frequency.

[2] Note that the cost of communication is ignored. This is true for applications that have no communications among processors. However, the simplified model does not limit the applicability of the algorithms presented in this section except Equation (18).

allocated power is shown. Then, the dynamic update of the power allocation and system parameters during run-time is described.

III.D.3.a. Initial Power Allocation Computation

To compute initial power allocation, the *weighted power usage function* (WPUF) is computed. The WPUF is a desired power allocation based on the event rate schedule and weight function.

$$WPUF(t) = u(t)\, w(t) \text{ for all } 0 \le t \le T$$

To maintain balance between externally supplied energy and dissipated power for a period, WPUF is multiplied by a constant as follows.

$$u_{new} = \frac{u(t)w(t)\int_0^T c(t)dt}{\int_0^T w(t)u(t)dt}$$

The surplus power function is

$$c(t) - u_{new}(t)$$

The integration of the above function shows the power available from rechargeable battery (not including externally supplied power) at time *t*.

$$P_{original}(t) = \int_0^t c(v) - u_{new}(v)dv$$

This function may exceed C_{max} or be less than C_{min}. If it exceeds C_{max}, then the externally supplied energy available cannot be charged to the battery, which results in the waste of available energy. If it is less than C_{min}, then computation may not be performed until the battery is recharged. Thus, if the power available at time *t* exceeds C_{max}, then it is desirable to dissipate some power before time *t* for useful tasks to obtain better energy utilization. If the power available at time *t* is less than C_{min}, then the power needs to be saved before time *t* by using less energy before time *t*. The adjustment of power allocation can be done using the following algorithm.

```
Algorithm 5: Power dissipation schedule adjustment
1   $S \leftarrow \{t | \frac{dP_{original}(t)}{dt} = 0$ and ($P_{original}(t) < C_{\min}$ or $P_{original}(t) > C_{\max}$)};
2   Sort $S$ based on $t$ in ascending order;
3   for all elements in $S$
4       if two consecutive elements, $t_0$ and $t_1$ satisfy $P_{original}(t_0) < C_{\min}$ and $P_{original}(t_1) < C_{\min}$ then
5           Remove an element that has larger $P_{original}(t)$;
6       if two consecutive elements, $t_0$ and $t_1$ satisfy $P_{original}(t_0) > C_{\max}$ and $P_{original}(t_1) > C_{\max}$ then
7           Remove an element that has smaller $P_{original}(t)$;
8   $t_0 \leftarrow$ the first element of $S$;
9   $temp \leftarrow t_0$;
10  $S \leftarrow S - \{t_0\}$;
11  while $S$ is not empty
12      t1 $\leftarrow$ the first element of $S$;
13      if $P_{original}(t_0) < C_{\min}$ and $P_{original}(t_1) > C_{\max}$ then
14          $P_{init}(t) = \frac{C_{\max} - C_{\min}}{P_{original}(t_1) - P_{original}(t_0)}(P_{original}(t) - P_{original}(t_0)) + C_{\min}, \quad t_0 \le t \le t_1$;
15      else if $P_{original}(t) > C_{\max}$ and $P_{original}(t_1) < C_{\min}$ then
16          $P_{init}(t) = \frac{C_{\max} - C_{\min}}{P_{original}(t_0) - P_{original}(t_1)}(P_{original}(t) - P_{original}(t_1)) + C_{\min}, \quad t_0 \le t \le t_1$;
17          $t_0 \leftarrow t_1$;
18          $S \leftarrow S - \{t_1\}$;
19  $t_1 \leftarrow temp$;
20  Adjust $P_{init}$ for $0 \le t < t_1, \quad t_0 \le t < T$ using Lines 13 – 16 by considering two segments as a
    one contiguous time segment $t_0 \le t < T + t_1$.
```

In lines 1–7, the algorithm first identifies the times at which the original power allocation reaches maximum or minimum. These times are used to adjust the power allocation. From lines 8 to 18, the algorithm adjusts the power allocation by allocating more power or less power based on this information. In the algorithm, the amount of stored energy depends on the original power allocation. However, other ways of adjusting can be used. For example, the power can be evenly distributed. From lines 19–20, the power allocation is adjusted between $0 \le t < t_1, \quad t_0 \le t < T$.

III.D.3.b. Initial Parameter Computation

For the given initial power allocation, the system parameters need to be computed. The status of the system at time t can be denoted using $(f_0, \ldots f_{n-1}, v_0, \ldots v_n)_t$, where f_i denotes the frequency of the processor i and v_i denotes the voltage of the processor i. When a processor is inactive, the processor is denoted using zero voltage or frequency. The frequency and voltage are such that $f_i \in F$ and $v_i \in V$, where F and V are sets of frequency values and voltages values, respectively. The number of active processors can be easily derived by counting processors that have non-zero frequency or voltage.

The parameter space can be reduced by the following observations: the first observation is that if the frequency of a processor is lower than f_0, where $f_0 = g(v_{\min})$ at time t, then changing the frequency will change the performance; however, changing the operating voltage does not change performance. The second observation is that if the frequency of a processor, f_1, is higher than or equal to f_0, where $f_0 = g(v_{\min})$, then the operating voltage to choose for the best performance/power ratio is $v_1 = g^{-1}(f_1)$. From these observations, the best operating voltage for a given frequency is determined by:

$$v = \begin{cases} g^{-1}(f), & if\ g^{-1}(f) \ge v_{\min} \\ v_{\min}, & if\ g^{-1}(f) < v_{\min} \end{cases}$$

Therefore, the system parameter space can be reduced to frequency only since the voltage can be derived from frequency using the above equation. Thus, in those systems, the system parameters can be represented using (f_0, … f_n).

Algorithm 6: Determining the number of processors and operating frequency

```
1    for all $n$ and $f$
2            Compute power-performance pair ($Power(n, f)$, $Perf(n, f)$);
3    for all ($n_v$, $f_v$) and ($n_w$, $f_w$), where $n_v \neq n_w$ and $f_v \neq f_w$
4            if $Power(n_v, f_v) \geq Power(n_w, f_w)$ and $Perf(n_v, f_v) <= Perf(n_w, f_w)$ then
5                delete ($Power(n_v, f_v)$, $Perf(n_v, f_v)$) pair;
6    $t$=0;
7        $E = \int_t^{t+\tau} P_{init}(t)dt$
8    ($n_t$, $f_t$)  $\leftarrow$ ($n_v$, $f_v$) of the power-performance pair of which $Power(n_v, f_v)$ best matches $E/\tau$;
9        $t = t + \tau$;
10       while ($t < T$)
11           Update $P_{init}$ using algorithm 3;
12           $E_{diff} = \int_t^{t+\tau} P_{init}(t)dt - \tau \min(P_{init}(t), P_{init}(t+\tau))$;
13           ($n_{tmp}$, $f_{tmp}$)    $\leftarrow$ ($n_v$, $f_v$) of the power-performance pair of which
                 $Power(n_v, f_v)$ best matches $\int_t^{t+\tau} P_{init}(t)dt / \tau$;
14           $Power_{overhead} = 0$;
16               $Power_{overhead} = Power_{overhead} + OH_n$;
17           if ($f_{t-\tau} \neq f_{tmp}$) then
18               $Power_{overhead} = Power_{overhead} + OH_f$;
19           if $E_{diff} > \tau\, Power_{overhead}$ then
20               ($n_t$, $f_t$)  $\leftarrow$ ($n_{tmp}$, $f_{tmp}$);
21           else then
22               ($n_t$, $f_t$)  $\leftarrow$ ($n_t - \tau$, $f_t - \tau$);
23           $t = t + \tau$;
```

In many systems, the same clock frequency is used for all processors. Thus, we can further simplify the problem for those systems by using the same frequency for all processors. Then, the parameter can be represented by (n, f).

If the parameter space is continuous and there is no overhead for changing frequency, voltage, and the number of processors, then the parameters can be computed as follows for two cases: (i) $f < g(v_{min})$ and (ii) $f \geq g(v_{min})$.

Algorithm 7 Dynamic update of the power allocation

```
1    $E_{diff} \leftarrow \int_{t-\tau}^{t} P_{init}(v) - P_{actual}(v)dv$
2    if $E_{diff} > 0$ then
3        Find time $w$ such that $P_{init}(v) = C_{\max}$;
4        for all $v$ such that $t \le v < w$
5            $P_{init}(v) = \min(E_{diff}\frac{P_{init}(v)}{\int_{t}^{w} P_{init}(v)dv}, C_{\max})$
6    if $E_{diff} < 0$ then
7        Find time $w$ such that $P_{init}(v) = C_{\min}$;
8        for all $v$ such that $t \le v < w$
9            $P_{init}(v) = E_{diff}\frac{P_{init}(v)}{\int_{t}^{w} P_{init}(v)dv}$
```

In real systems, it is obvious that the number of processors cannot change continuously. Also, in many systems, the frequency can be chosen from a set of pre-selected frequency values, and there is an overhead for changing parameters. For example, when a processor is changed to active mode from stand-by mode, it needs to be initialized before it can process tasks. Let us denote the overhead due to the change in the number of processors and frequency change as OH_n and OH_f, respectively. We assume that the system parameters can be changed at time $t = i\tau$, $i = 0, 1, 2, \ldots, T/\tau-1$, where τ is a constant. Algorithm 6 computes the system parameters so that the power usage closely follows the allocated power schedule while obtaining the best performance/power ratio.

Lines 1 - 2 compute the parameter-power pair table. Lines 3–5 check the table and if a pair shows less performance with the same power, then the pair that shows less performance is removed. Lines 6–9 calculate the parameter for time 0. In lines 10–22, the parameters for the rest of the periods are computed. In line 11, the P_{init} is recomputed to accommodate the difference of the initial power allocation and the power usage due to the discrete nature of the computed parameters. Lines 12–13 compute the new parameters. Lines 14–22 compute the overhead due to the change of the number of processors and frequency. If they are worth changing, then the parameters are changed; otherwise, the current parameters are maintained.

In Algorithm 7, a deviation during initial system parameter computation due to the discrete parameter space is accommodated. The deviation is computed after each interval τ. If less energy was used than planned, then energy is allocated to the next computations. If more energy was dissipated than expected, then less power is allocated to the next computations.

III.D.3.c. Simulation Results: An Example System

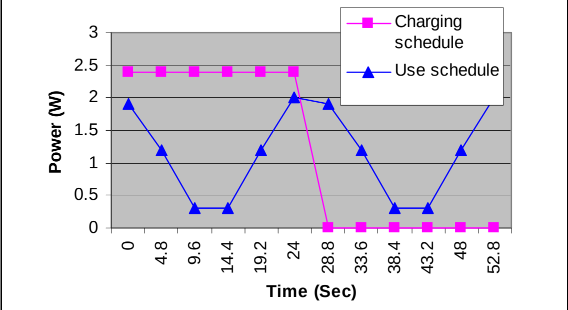
We implemented a simplified FORTE (Fast On-Orbit Recording of Transient Events) [1] signal processing application that detects radio frequency events from a satellite. In FORTE, when the signals from sensors trigger an analogue threshold circuit, digital signal processing is performed to check if it has the characteristics of an interesting RF event. Most of the computation for the system is an FFT that takes about 60% of the execution time. In this implementation, we used only the FFT portion of the application. Since the simulated platform does not support floating-point operations, we implemented fixed-point FFT operations.

We used one of the processors as a controller processor and the rest of them are used for FORTE signal processing. The controller processor computes the P_{init} and updates it. Based on P_{init}, it sends frequency and active/stand-by mode change commands to other processors. Each processor checks the command from controller processor after each computation.

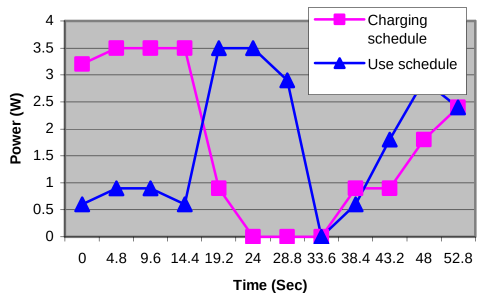
In our experiments, we ran the M32R/D processor at between 20 MHz and 80 MHz. In this simulation, we set the possible frequency choices to 0, 20, 40, and 80 MHz. We set the voltage (3.3 V) to $v_{min} = v_{max}$.

The measurement of the execution time for the 2K sample FFT at 20 MHz was 4.8 sec. Thus, we set the τ as 4.8 sec. *T* is set to 57.6 sec. Thus, there are 12 parameter updates in a period.

The run-time trace of the P_{init} is simulated for two scenarios. Figure 14 (a) and Figure 14 (b) show charging schedules and power usage schedules for two different scenarios. The detailed simulation results are found in [11]. In both scenarios, after five iterations, the integration of the initial power allocation is ore than the minimum requirement (0.098). In this simulation, we assumed no overhead for changing the number of processors and frequency.



(a) Charging and use schedule for Scenario I



(b) Charging and use schedule for Scenario II

Figure 14. Simulation results

Two metrics are computed for comparison: *energy wasted* and *undersupplied energy*. The energy wasted is energy that was not used for useful computation. This happens when the battery is fully charged while the external source has energy to supply. Undersupplied energy means the energy needed for computation but not available at that time. For comparison, we implemented a static algorithm. Since no overhead for changing the number of processors or frequency is assumed, the system is turned off while

Table 15. Comparison of algorithms

Algorithm	Metric	Scenario I	Scenario II
Proposed	Wasted energy	13.68 J	6.18 J
	Undersupplied energy	23.11 J	6.27 J
Static	Wasted energy	40.93 J	69.33 J
	Undersupplied energy	39.33 J	67.91 J

there is no input data to process. If the externally supplied energy is more than the usage, then the difference is charged to a rechargeable battery. If more energy is used than supplied energy, then the difference is supplied from battery.

The results are shown in Table 15. The results show that the proposed algorithm reduces the wasted energy up to ten times for Scenario II when it is compared with the Static algorithm. Also, since it allocates less power if it anticipates a situation when the energy is undersupplied, it lowers the probability of the undersupplied situation.

IV. Application Algorithm-Level Power Management

For this work, we have focused on power-aware processing for a remote-sensing application similar in nature to the mission of FORTÉ. The FORTÉ satellite was launched in August of 1997 and carries a suite of instruments used for studying the optical and RF signals from lightning in the Earth's atmosphere. The results from FORTÉ have led to a better understanding of the relationship between optical and RF lightning events, and future satellite missions can use this knowledge to help provide global lightning and severe-storm monitoring [1]. The processing algorithms for the RF lightning signals have been chosen in this study.

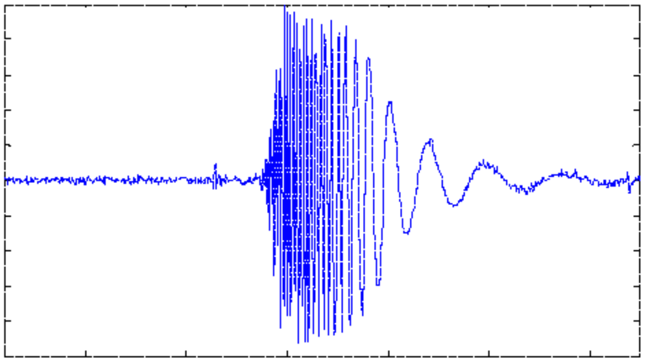


Figure 15. Time vs. amplitude profile of chirp

IV.A. Ionospheric-Dispersed Signals

A RF lightning event in the Earth's atmosphere generates a dispersed signal in the VHF spectral region. The dispersion is such that low frequencies of the signal are delayed, as it propagates through the ionosphere. This is known as a "chirp" signal. A simulated chirp signal is shown by the graph in Figure 15. The graph is an illustration of the time-domain signal where frequency decreases with increased time. The time taken for a given frequency of the chirp signal to arrive at the on-orbit receiver is related to the total electron content (TEC) of the ionosphere along the direction of the signal travel, the given frequency, and the signal time-of-arrival if ionospheric dispersion did not exist [14]. The total electron content (TEC) represents the number of electrons in a unit-area cross-section of an ionospheric column along the signal path. This atmospheric property is related to the propagation of radio signals through the ionosphere that can distort or bend the signals over the horizon. TEC is also related to the surface temperature of the Earth, and thus, could be viewed as an indicator for storm severity.

IV.B. Algorithms

Four algorithms, which are shown in Figure 16, are available to operate on the data in order to yield an estimate of the TEC value. The four algorithms will be referred to as: 1) least-mean-squares (LMS), 2) maximum-likelihood (ML), 3) software trigger box (ST), and 4) match-filter bank (MF). Using only the time-frequency data pairs provided by the channels of the analog trigger box, the first two algorithms use curve-fitting techniques. While the LMS is a deterministic algorithm, the ML is forced to be deterministic by only allowing for 20 iterations to be performed [15]. The remaining two algorithms make use of the 2048-sample, 12-bit, digitized waveform data. Using digitized waveform data, the software trigger box algorithm utilizes frequency domain processing to provide an estimate of TEC. The software trigger box algorithms first transforms the time-domain data into a non-overlap spectrogram composed of boxcar-windowed, 32-

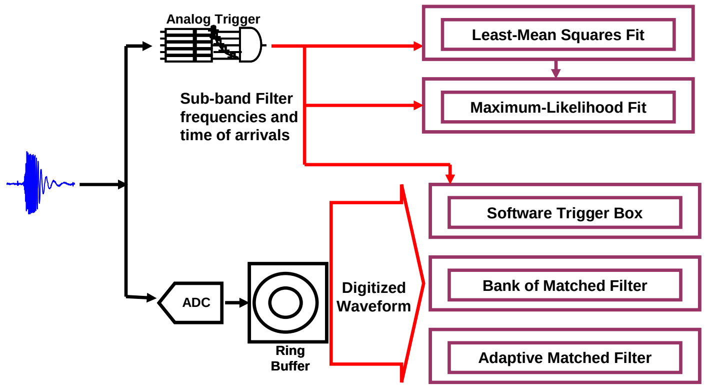


Figure 16. Four Forte data processing algorithms

sample FFTs. Upon determining the associated time indexes for the maximums of each of the seventeen non-negative frequency bins, a maximum likelihood algorithm is performed on the data pairs constructed from the bin-maximum times and the center frequencies of the FFT bins. The MF algorithm also utilizes frequency-domain processing. By generating simulated exemplar time-domain waveforms of different TEC and transforming them into the frequency domain, a bank of matched filters can be constructed that spans the space of possible TEC values. A correlation peak is rendered by performing a fast correlation algorithm on the waveform data and a TEC-specific filter. Exploration of the match filter bank for the greatest correlation-peak value is done with a "focus-in" decision tree so that only ten fast correlations are performed to yield an estimate of TEC. However, since the value of the winning peak is not quantized or hard constrained like the TEC estimate, this peak value will be used for post-processing detection work in this paper for the match-filter bank algorithm.

IV.C. Power Measurements

Power usage measurements for the four algorithms were obtained through experiments conducted on a 266-MHz PowerPC 750 microprocessor running the VxWorks™ operating system. Both time-to-execute values and power usage estimates (RMS and peak current) were determined for the PowerPC 750. The time-to-execute values are average values over a test set of 21 trials cycled 20 to 100 times. Each trial

Table 16. Power management for Power PC 750

Algorithm	Current (amps -peak)	Execution Time	Energy (Joules)
Least Mean Squares	2.06	3.4 us	18.7e-6
Maximum Likelihood	2.06	183 us	1.02e-3
Software Trigger Box	2.18	8.34 ms	47.3e-3
Match Filter Bank	2.04	470 ms	2.35

used synthetically generated data that simulated a chirp-signal event being received by a space-based receiver system containing an analog trigger box and a waveform digitizer.

Power usage for the PowerPC 750 executing the benchmaking code is presented in Table 16. The Jet Propulsion Laboratory (JPL) power-aware testbed consists of a Wind River PPC750 266-MHz processor board that is running VxWorks 5.4.2. The processor operates at a constant 2.67V and current consumption is measured with a Tektronix TDS 7104 Digital Phosphor Oscilloscope. Current is sampled with the Tektronix TCP202 probe that is wired to the board. Software compilation is done with a VxWorks Tornado 2.0.2 programming tools which uses the GNU C compiler. The software is compiled and downloaded to the testbed with the Tornado target server shell. The programs are run until an "average" current signal snapshot is taken with the oscilloscope. The "average" signal is determined manually by watching the current response during several program runs. The snapshot is taken when the current response produces a fairly consistent signal and consistent measurement value.

IV.D. Post-Processing Detection

The output of the four algorithms can be compared against unique thresholds to determine if a false alarm has been generated by the analog trigger box. Figure 17 shows the concept expressed in terms of its effect on the ROC curve. Of course, lost detections can not be corrected for in post processing since the analog trigger box cues the collection of data and the execution of the algorithms. Unique thresholds for each algorithm are needed since the algorithms arrive at their results differently and require different amounts of power to obtain those results.

IV.E. Validation Experiment

In order to validate the concept of using the parameter estimation algorithms to improve detection performance, another hundred thousand trial Monte Carlo experiment was performed. This experiment made use of the calculated thresholds to test the detection validity of each trial. Of course since the true nature of each signal is known, detection performance numbers can be determined for each algorithm. The results of the experiment are shown in Table 17. The experiment result for the probability of detection of the analog trigger box was 54.86% and its probability of false alarm was 7.25%.

Table 17. Monte Carlo results for detection performance

Algorithm	Probability of Detection	Probability of False Alarm
Least Mean Squares	50.75%	3.58%
Maximum Likelihood	50.75%	3.74%
Software Trigger Box	52.60 %	2.01%
Matched Filter Bank	50.79%	0 %

IV.F. Results

From the results of the validation experiment, a difference between the four algorithms is presented. False alarms are reduced most dramatically by the match-filter bank algorithm. They were reduced to a level below what could be determined by the Monte Carlo experiment where 50-thousand opportunities were presented. The software trigger box algorithm provided the second greatest reduction in false alarms by nearly a factor of 4 better than the analog trigger box. The LMS (least-mean squares) and ML (maximum likelihood) algorithms performed approximately the same and provide the least improvement of the four algorithms. Roughly a factor of two decrease in probability of false alarm is achieved by the post-processing of the results of these two parameter estimation algorithms. The correlation of these results with each algorithms energy usages is presented in Figure 16. A general relationship can be stated that the more energy that is expended, the greater the reduction in the probability of false alarm. Ignoring the ML-labeled point, the graph shows that several orders in magnitude of energy (expended) are needed for a meaningful improvement in probability of false alarm. However, it does show that with enough energy, the false alarm probability can be made zero (or close to zero). Detailed results and discussions are presented in above.

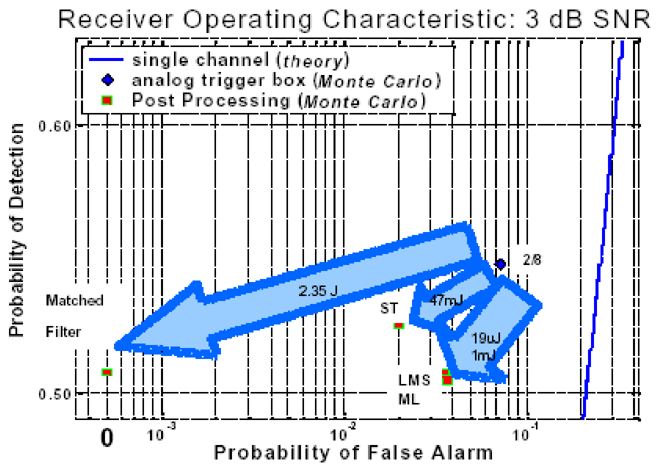


Figure 17. Detection performance improvement shown on ROC curve with associated energy usage

V. Experimental Results on PAMA Implementations

The results of experiments performed on the three PAMA architecture implementations are presented in this section.

V.A. SLIIC-QL Board

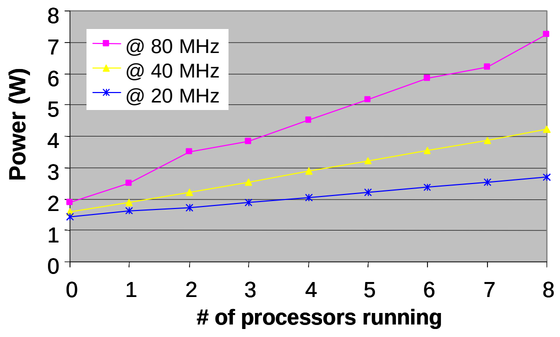


Figure 18. Power consumption for SLIIC-QL board

The SLIIC QL board has eight Mitsubishi M32R/D processors in above. Each processor has dynamic range of clock frequency between 20 MHz and 80 MHz with a peak performance of 80 MIPS. Peak power consumption of each processor is 1.44 Watts, and the power consumption of eight processors is 7.24 Watts. We ran the Forte application from Los Alamos, which was described previously, and changed the number of active processors and clock frequencies. The results are summarized in Figure 17. Power consumption increases linearly as the number of processors increases. There is some amount of leakage current even when no computation is used. When no computation is done, about 1.5 Watt is consumed by the SLIIC-QL board.

We implemented two types of networks using the network FPGA: a ring and a crossbar. We conjectured that the power consumption of the crossbar should be always higher than the ring network, however, it turns out that power consumption depends more on the implementation logic than the pure topology. Our implementation of the ring network is more complex than the crossbar network due to the buffering of control signals, so the ring consumed more energy than the crossbar network, as shown in Figure 19. We also observed that actual power consumption of the network depends significantly on the amount of communication that an application requires.

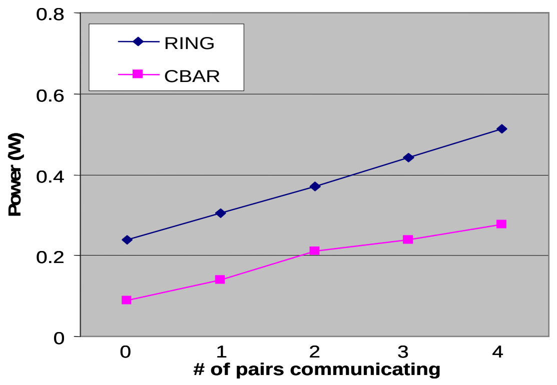


Figure 19. Power consumption comparison between ring and crossbar networks

V.B. SH-4 Based PAMA

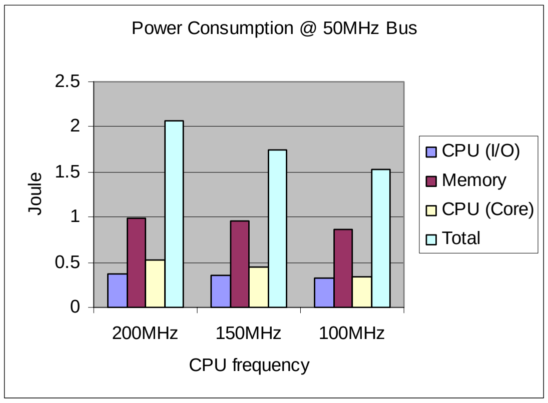
We performed experiments to see the changes in power and energy consumption: 1) as CPU clock frequency changes 2) as the memory bus frequency changes 3) as CPU voltage changes and 4) as different algorithms are used.

V.B.1. CPU Frequency Scaling

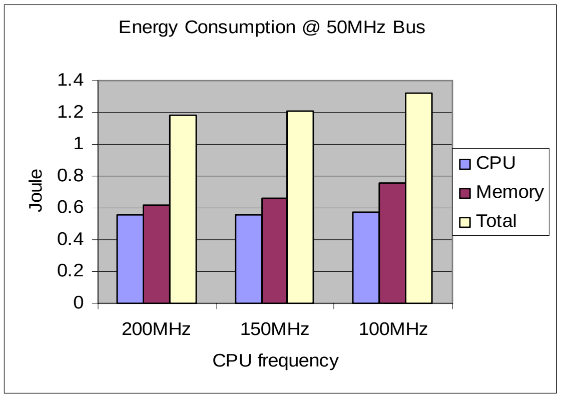
We changed CPU clock frequencies while bus frequency and CPU voltage are fixed. We ran the LMS, ML, ST, and MF algorithms from the FORTE application. For the four algorithms, we observed a similar trend in terms of power consumption and energy consumption. As CPU clock frequency decreases, power consumption decreases. However, energy consumption increases because the execution time increases more than the power consumption decreases as clock frequency increases. The result is shown in Figure 20.

V.B.2. Frequency Bus Scaling

We also changed memory bus clock frequencies while the CPU clock frequency and CPU voltage were fixed. We ran the LMS, ML, ST, and MF algorithms of the FORTE application. The number of memory accesses in an application affects execution time and energy consumption. In terms of memory usage, both LMS and ML perform the fewest memory options, and are CPU intensive. ML is more memory intensive, and ST is in the middle. We can predict that the more memory accesses an application performs, the longer the application takes to run at lower bus frequencies. This longer execution time results in more energy consumption. This prediction was validated for the LMS, ML, and MF algorithms as is shown in Figure 21. The lowest energy consumption is found at the fastest bus frequency in Figure 22 for MF, which is a memory intensive algorithm. In the case of ST, the lowest energy consumption is found in the middle. The result is shown in Figure 22.

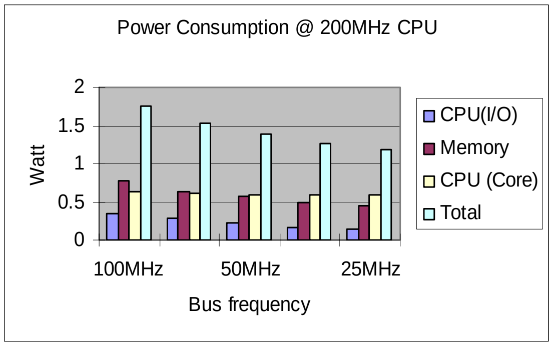


(a) Power Consumption Change

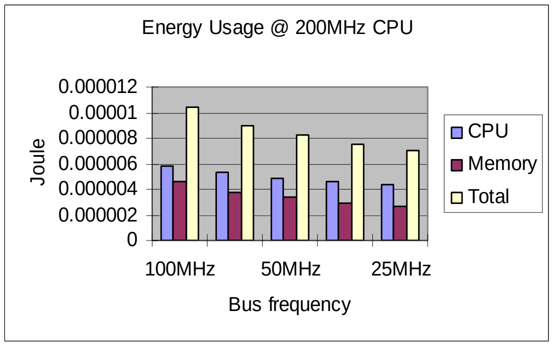


(b) Energy Consumption Change

Figure 20. Power and energy consumption as a function of CPU clock frequency

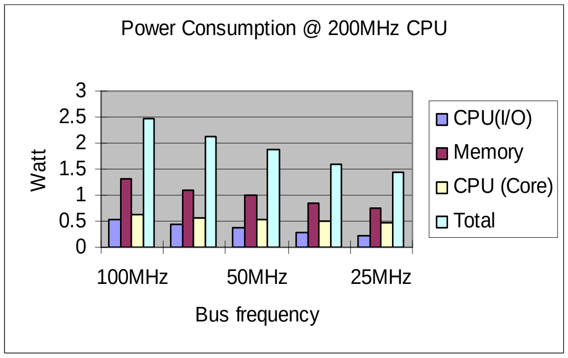


(a) Power Consumption

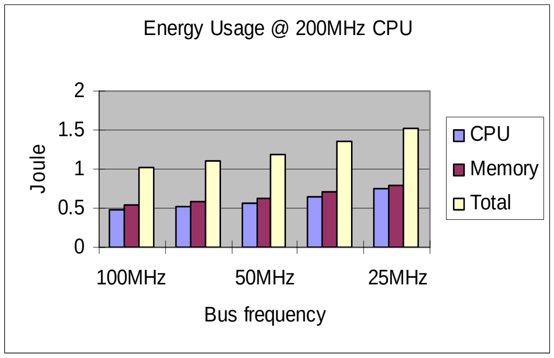


(b) Energy Consumption

Figure 21. Power and energy consumption as a function of bus frequency for LMS algorithm

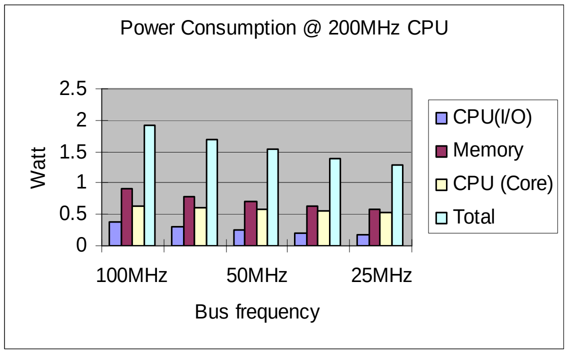


(a) Power Consumption

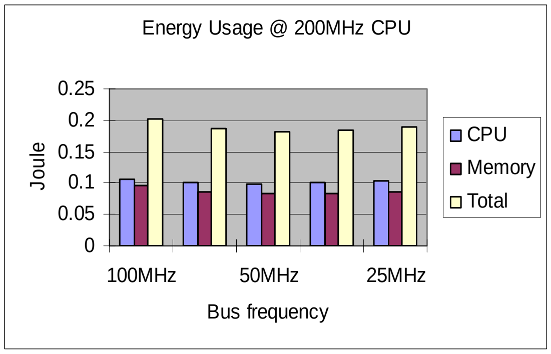


(b) Energy Consumption

Figure 22. Power and energy consumption as a function of bus frequency for the MF algorithm



(a) Power Consumption



(b) Energy Consumption

Figure 23. Power and energy consumption as a function of bus frequency for the ST algorithm

V.B.3. Voltage Scaling

The Hitachi SH-4 is not advertised as a voltage-scalable processor, however, we carried out voltage scaling and found the lowest working voltage for each frequency. The summary of our experiment is shown in Figure 23. As voltage decreases, power consumption of CPU becomes smaller and power consumption of the memory becomes more significant. The changes in power consumption for the MF algorithm at three different voltages are shown in Figure 23. As shown, the power consumed by memory becomes more dominant as CPU voltage becomes lower. Therefore, voltage scaling for memory chips is an important area for future research.

V.B.4. Multiprocessor Power Consumption

Table 18. Power as a function of voltage for PAMA SH-4 node

CPU frequency	Voltage	Power consumption (system idle)
200 MHz	1.80V	0.66 Watt
150 MHz	1.45V	0.50 Watt
100 MHz	1.30V	0.39 Watt

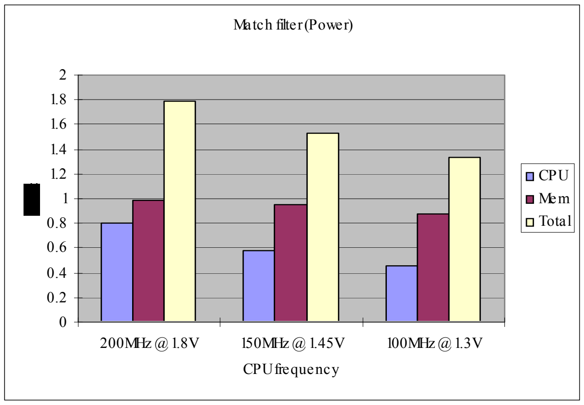


Figure 24. Power consumption with voltage scaling for SH-4

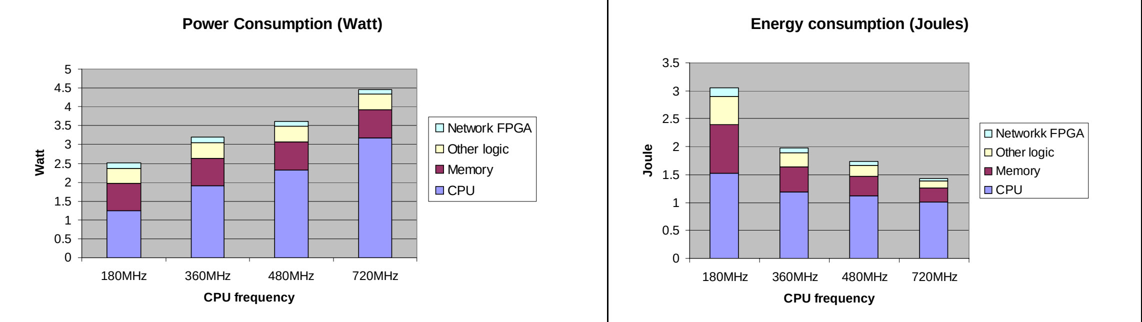
By changing the number of processors and their clock frequencies and voltages, we can achieve a bigger range of power consumption. The maximum power consumption of the Hitachi SH-4 processor in standby mode is 11.16 mW. The measured power consumption of the memory in self-refresh mode is 45.8 mW. Therefore, the minimum power consumption of single PAMA-SH node is about 57 mW. Power consumption for the MF algorithm at 200MHz CPU frequency is 1.8 W. So the power consumption of the MF algorithm running on four processors can be between 171 mW and 6.8 W, depending on event rates.

V.C. PowerPC 750 FX-Based PAMA

We performed experiments on the PAMA-PPC architecture to see the changes in power and energy consumption 1) as CPU clock frequency changes 2) as bus frequency changes 3) as CPU voltage changes.

V.C.1. CPU Frequency Scaling

We changed CPU clock frequencies while bus frequency and CPU voltage are fixed. We ran the MF algorithm, which is the most CPU intensive among the four algorithms of the FORTE application. As CPU clock frequency decreases, power consumption decreases. However, energy consumption increases as seen for the SH-4. Unlike the SH-4, power consumption by the PowerPC 750-FX is dominant because of its high clock frequency. The results are shown in Figure 24.

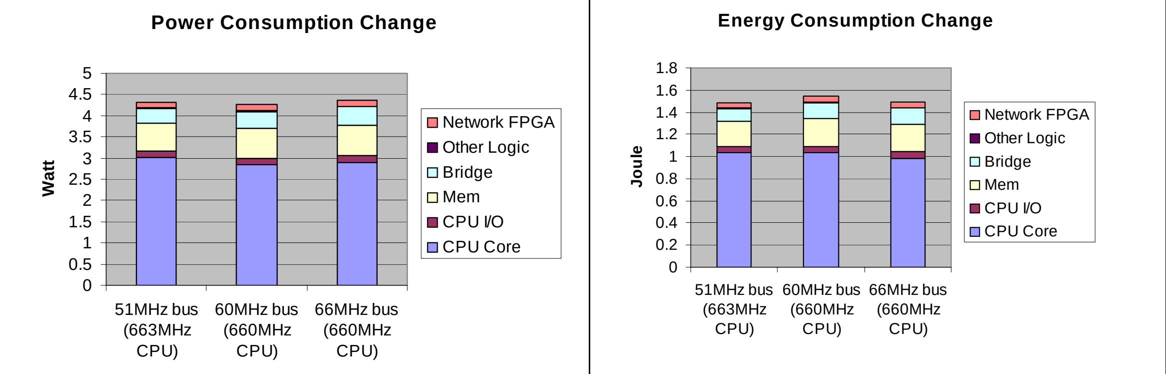


(a) Power Consumption (b) Energy Consumption

Figure 25. Power and energy consumption as a function of CPU for PowerPC 750-FX

V.C.2. Bus Frequency Scaling

We changed memory bus clock frequencies while the CPU clock frequency and voltage are fixed. We ran the MF algorithm of the FORTE application with a range of bus frequency between 50MHz and 66MHz. The results are shown in Figure 25. Memory bus frequency does not have a significant effect on power consumption for the PowerPC 750-FX-based nodes.



(a) Power Consumption (b) Energy Consumption

Figure 26. Power and energy consumption of PAMA-PPC at different bus frequencies

V.C.3. Voltage Scaling

Like the SH-4, the IBM PowerPC 750FX is not advertised as a voltage-scalable processor. However, we carried out voltage scaling within the range supported by the programmable logic on the network card, however, the voltage scaling range that the network card can provide for the 750FX is only between 1.3V and 1.4V, which is too narrow to notice show significant power savings.

V.C.4. CPU Mode

The IBM PowerPC 750FX has four modes: Run, Doze, Idle, and Sleep. Power measurements did not show a significant difference between Doze and Idle modes. The comparison of power consumption between Run, Idle and Sleep modes is shown in Figure 26. The power consumption is smallest in Sleep mode, and it is the largest in Run mode at 720MHz. Dynamic range of power consumption is between 1 W and 5 W.

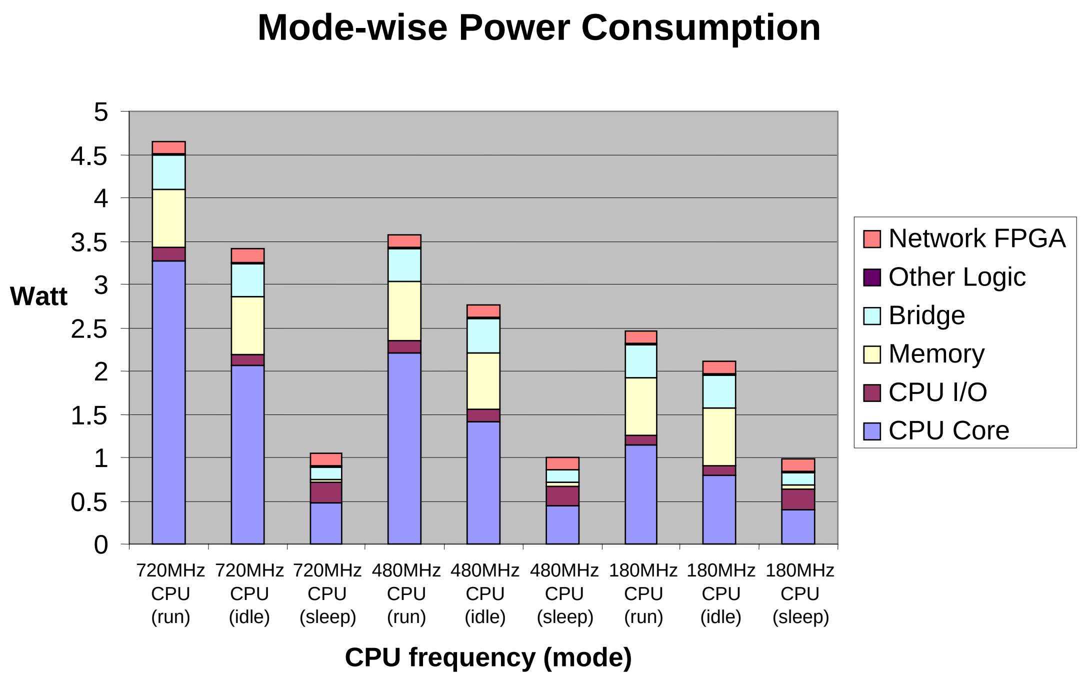
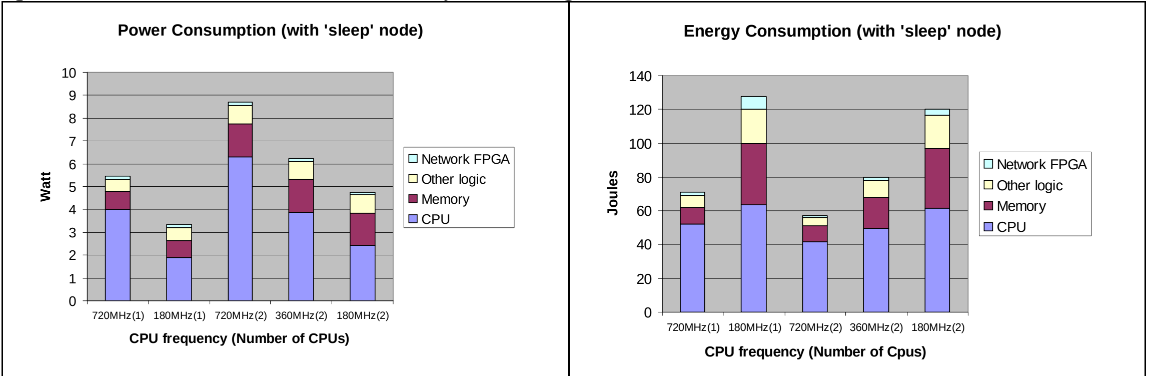


Figure 27. Power consumption at different CPU frequencies and modes

V.C.5. Multiprocessor Experiment

We changed the parameters (CPU frequency, number of nodes) and compared power and energy consumptions of those configurations. The CPU frequencies used are 180MHz, 360MHz, and 720MHz. The number of processors varies from 1 to 2, because we have only two working nodes. When a node is not involved in the computation, it is put into Sleep mode, in which the processor board consumes 0.9 W. Dynamic range of power consumption is between 3.2 W and 8.6 W for two nodes. From the data, if four processors were used, we estimate the dynamic range would be between 5.0 W and 17.2 W.



(a) Power Consumption

(b) Energy Consumption

Figure 28. Multiprocessor experiment results

VI. STRETCH

VI.A. Brief Project Summary

STRETCH was a DARPA-funded seedling effort at ISI-East and Virginia Tech to investigate concepts in e-textiles as they apply to large-scale sensor systems. STRETCH was funded through the PAMA cooperative agreement. The experience of the principal investigators enables them to focus the STRETCH effort on large-scale acoustic beam-forming arrays, and in particular, the information technology aspects that are unique to the e-textiles domain in such an application [16][17][18].

VI.B. Project Overview

The research at Virginia Tech concentrated on the design and construction of a series of e-textile prototypes as well as on the design and implementation of a simulation environment to support exploration of the e-textile design space. This research was primarily focused on large-scale woven textiles to be used as sensor arrays. Within sensor arrays, the emphasis was on acoustic beam-forming for the purpose of detecting and locating heavy vehicles.

Prototype development focused on the incorporation of sensors, actuators, communications, and computation into a woven fabric. The development process culminated in the construction of a thirty-foot long woven fabric that incorporated twenty-eight microphones, four signal-processing units, and an intra-fabric communication system. Each of the four microphone clusters in this prototype forms a beam, which indicates the direction of arrival of an acoustic signal, then communicates with the other clusters to cooperatively improve the beam and determine the location of the acoustic emitter (vehicle). The photograph in Figure 29 was taken during an outdoor test of this large-scale prototype.

The goal in the modeling and simulation environment portion of the project was to design and implement an environment useful for the exploration of the e-textile design space. To make this problem tractable and keep the environment grounded in reality, we tightly coupled the development of this environment to our prototype development efforts. The resulting environment, described in detail later in the report, is capable of exploring a wide range of design options, including the number and position of sensors, the number of processing units, a range of acoustic signal types, the size/shape of the fabric, the communication topology, and the beam-forming algorithm. 29 shows a prototype e-textile developed under this effort.

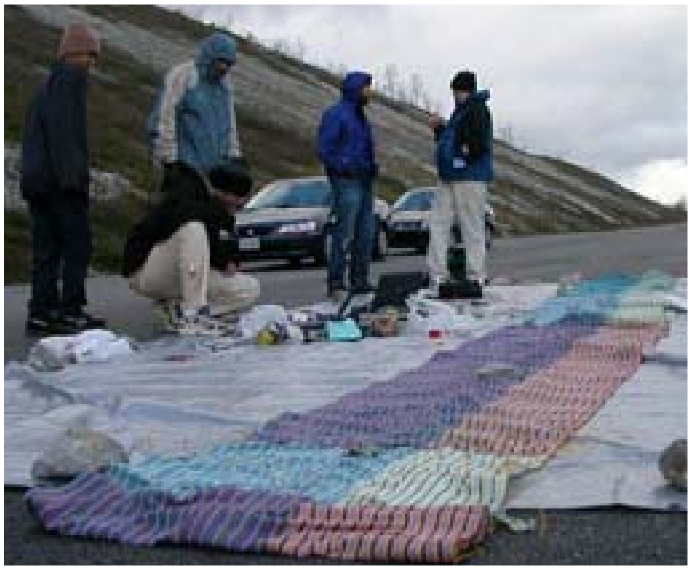


Figure 29. Thirty-foot prototype textile deployed for an outdoor test

In the remainder of this report, we describe the motivation for e-textiles, the prototype construction results, the modeling and simulation environment, and the conclusions reached as a result of this research.

VI.C. Why E-Textiles?

The advantages of e-textile-based large-scale sensor networks include:

- **Flexibility**: textiles are inherently flexible and strong, allowing them to be stored conveniently and to be deployed easily.
- **Affordability**: extensive low-cost manufacturing processes exist for textile fabrication.

- **Efficiency**: sensor networks require extensive communication to cooperatively form results, wired communication within the textiles offers a large power consumption and component cost benefits as compared to wireless communication.
- **Longevity**: the large surface area of textiles allows for the incorporation of multiple redundant components for fault tolerance as well as extensive energy harvesting and storage.
- **Familiarity**: textiles permeate our everyday environment; users are familiar with them and know how to operate them. Augmenting familiar objects with new functionality will result in a shorter learning curve and wider acceptance.

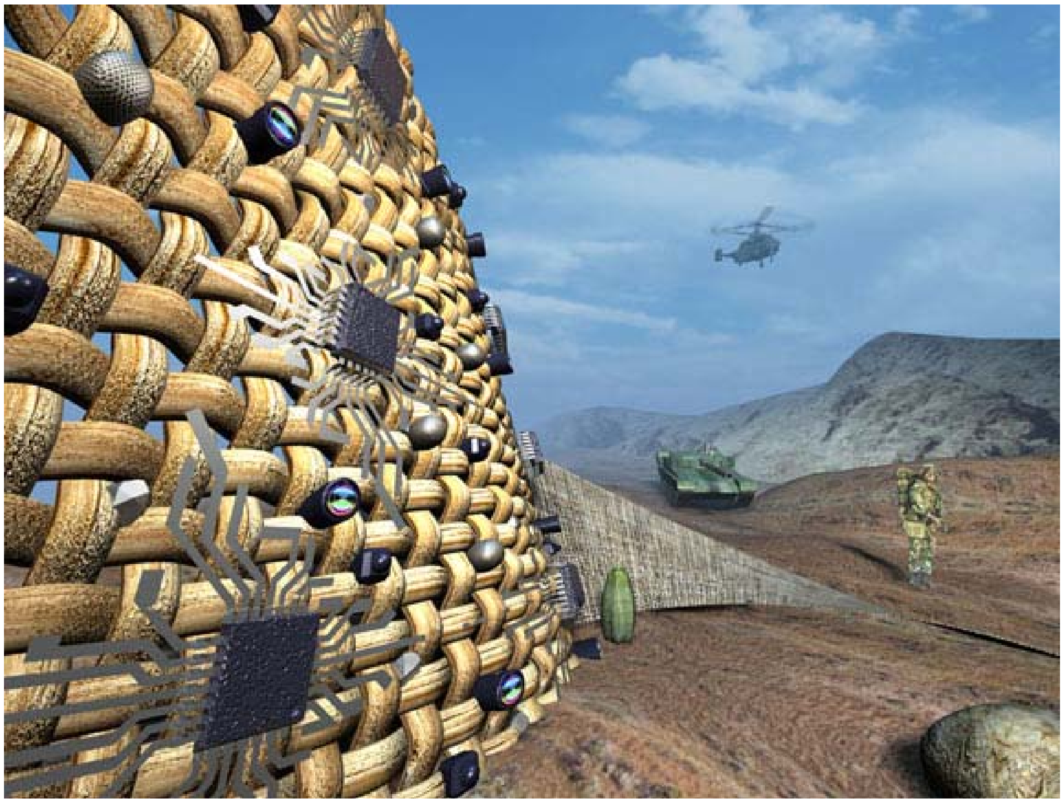


Figure 30. Conceptual view of an e-textile deployed for battlefield sensing

Our philosophy in this project has been to keep these advantages in the foreground during the design process, while realizing that this is a nascent technology with many outstanding research issues. The beamforming textile is ultimately envisioned for battlefield deployment as a parachute, camouflage netting, or some other already extant textile (see Figure 30). Our thirty-foot prototype textile is clearly not in a military form factor. It was however, designed for redundancy, power-aware operation, and affordability.

E-textiles offer an excellent platform for the integration of heterogeneous components. In the case of woven textiles, our focus, components in a fiber form can be woven into the textiles, while discrete components can be attached. The large surface area allows for a large number of components to be integrated; this allows for fault tolerance as well as the opportunity to select the best sensor for the current task.

VI.D. Prototypes

The motivation for the development of prototypes was to allow us to understand the issues involved in the design of e-textiles, to provide us with an experimental platform, and to demonstrate that we could develop a useful large-scale textile sensor array. Satisfying these goals required overcoming several hurdles through a series of small prototypes. Unfortunately, it is not possible to order a prototyping development kit for e-textiles. In addition, as of the time we began this work, little research on the attachment and integration of discrete electronic components to fabrics had been performed.

To gain a better understanding of textile manufacturing, Mark Jones visited A.M. Seyam at his facility in the NC State College of Textiles. To construct our series of prototypes, we have worked with a local weaver on a contractual basis, keeping the large-scale manufacturing processes in mind. In this project, we were primarily interested in prototypes that were comprised of discrete components attached to a fabric; the fabric itself has metallic fibers at prescribed locations that allow for components to access power and ground as well as communicate amongst themselves (see Figure 31).

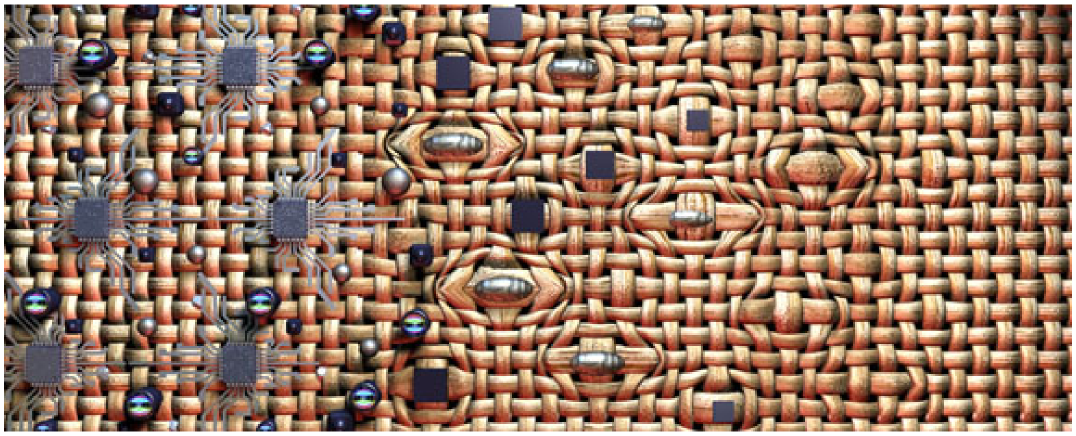


Figure 31. Conceptual integration of discrete sensors and processing elements on a textile

Our first series of prototypes investigated the attachment issues for different components, including discrete microphones, sensor fibers, and actuator fibers. We were also concerned with the problem of making/avoiding connections between wires in the fabric. Several prototypes, shown in Figure 32, were constructed. From these prototypes we drew the following conclusions:

- The spacing of metallic fibers within the fabric must match the pitch of the pins of components to be attached. The desired spacing has implications for both the metallic and non-metallic fibers chosen. It is difficult to achieve extremely tight spacing and to attach components with high precision; components requiring a large number of connections will be difficult and expensive to attach to fabric. The spacing must be such that each of the pins (or other attachment device) on the component touch the metallic fiber they are to mate with and no other metallic fiber. The precision required is driven by the size of the fibers, the size of the pins, and the space between the pins. This drives the decision to keep wide spacings on e-textiles, relative to pin spacings on a typical integrated circuit. We have achieved a spacing of less than one cm on our e-textiles prototype.
- A multilayer woven fabric can be effectively used to insulate wires in the x and y directions while making selective connections between the layers. This process allows uninsulated wires to be used. The disadvantage of this approach is an increase in the raw material cost as well as a heavier resulting textile. The raw material cost and weight will increase in proportion to the number of layers.
- The durability of the e-textile under repeated flex conditions is highly dependent on the nature of the metallic fibers. Single-core wire, for example, will have a very short lifetime. Perhaps the best candidate is a 100% stainless steel fiber made by Bekintex that is constructed of many extremely thin fibers, giving a very long lifetime along with the other positive properties of stainless steel in a harsh environment. PlasticsOne, the company that constructed several e-textile connectors for Foster-Miller, reported to us in conversation that a specially constructed copper wire also survived repeated flex well.
- Careful use of conductive epoxy can be used to make connections between wires and between components and wires. A similar process was described to us by Triton Systems based on work they performed on a DARPA SBIR (Small Business Innovation Research).
- Connectors attached to the fabric can be used to then attach discrete components. We have investigated a range of connectors and use them in our final prototypes. We have also considered

the “button” approach that Luna Innovations has also described to us based on work they performed on a DARPA SBIR.

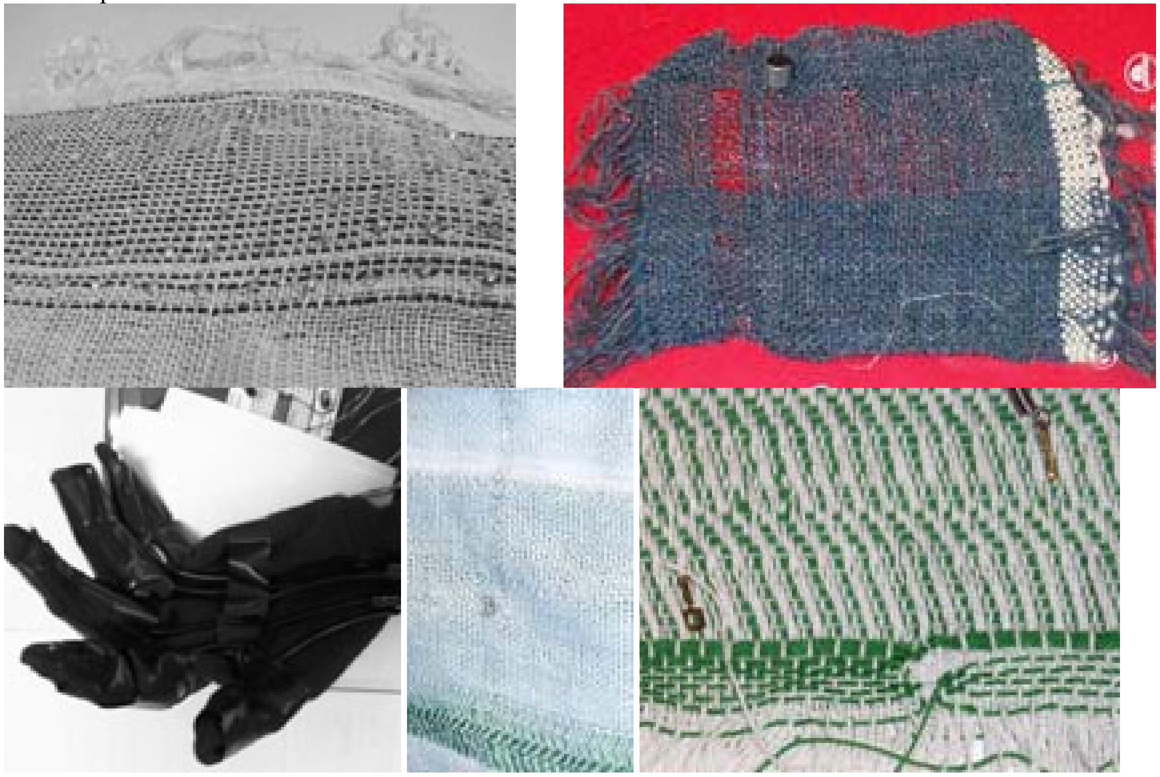


Figure 32. (Counter-clockwise form upper left) LED array attached with conductive epoxy, discrete microphone to fabric, shape memory alloy woven into fabric, conductive epoxy via, and piezoelectric fibers on glove

Based on this preliminary prototype work, we constructed two beam-forming prototypes (see Figure 33) that are each approximately 32” x 32”, each with a single cluster of seven discrete microphones attached to a signal-processing unit. The prototypes have wires woven into the textile and connectors attached to these wires at appropriate points. The signal-processing unit is a printed circuit board developed by Virginia Tech specifically for this project. The signal-processing unit contains analog

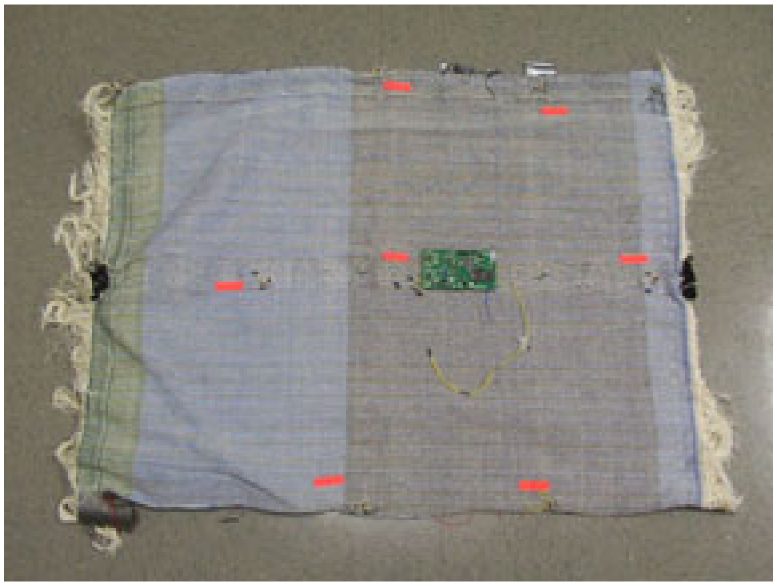


Figure 33. E-textile prototype

circuits for amplifying and filtering the microphone signals, an A/D unit for converting these signals to digital form, and an Analog Devices low-power DSP for performing the beam-forming algorithm. The board is attached to the fabric via a set of pins that mate to a connector attached to the fabric. This board serves as a carrier for the discrete components; the components on the board have hundreds of pins while the board itself only carries a small number of pins to the fabric. These pins are used to carry power, connect to microphones, and communicate with other boards through the fabric. This board is the prototype of what aggressive packaging techniques could turn into a very small unit. The board can sample and process data from all seven microphones at approximately 1000 samples per second in real-time.

The culmination of our prototyping efforts is a thirty-foot long, four-cluster textile. Each cluster has seven microphones and communicates with its neighbors to collaboratively compute the location of a vehicle. The textile is able to operate at an extremely low power because each cluster enters a low-power sleep mode and awakens only upon receiving a strong acoustic signal or a communication from another cluster. We have run the entire prototype for two days on a single nine-volt battery.

VI.E. Modeling and Simulation

To allow for future generalization, we have decomposed our modeling and simulation environment design and implementation process into three components, physics, architecture, and application software as described in Figure 34. The physics component models the behavior of the activity to be sensed; in this effort, we modeling acoustic propagation of signals in the 100-400 Hz range. The architecture reflects the e-textile structure, including the number/type/location of sensors and processing units and can easily be changed to allow investigation of different architectures. The application software runs on the processing elements and can be changed to allow for different algorithms and communication paradigms.

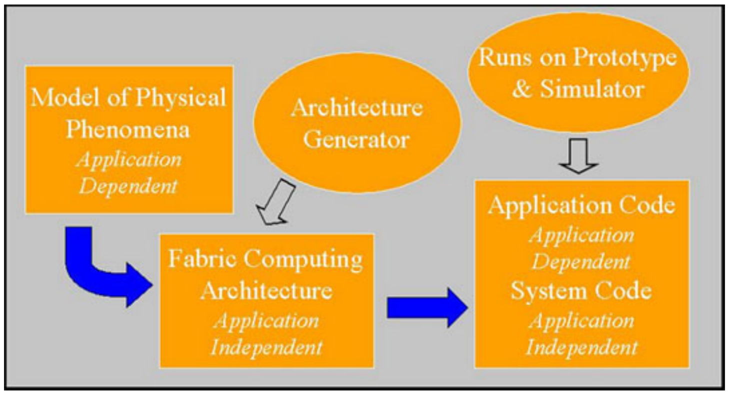


Figure 34. E-textile processing design flow

This structure allows the system to be adapted to new problems by changing the physics component and allows for architectures to be changed independently of other components. We have leveraged this capability during development when we move code between simulation and the prototype. The application software component allows users to write C code that can execute in the simulation environment and later easily ported to the textile prototype environment.

Because of its widespread use and open source, we chose Ptolemy II as the basic simulation environment. Figure 35 is a Ptolemy II screen shot of the acoustic beam-forming simulation environment. This environment has, in combination with measurements from prototypes, been used to model the power consumption and computed location accuracy for a range of e-textile architecture configurations. Specifically, we have varied the number of clusters, the number of microphones per cluster, the sampling rate, and the sample sizes using a fixed data set and evaluated the power consumption and compute location accuracy. These architectures were compared against the performance of a wireless sensor network (evaluated through experimental measurement and simulation); the e-textile network was found to deliver the same accuracy at the cost of much lower power consumption (due to the lower power consumption of communicating through wires in the textile compared to communicating through wireless).

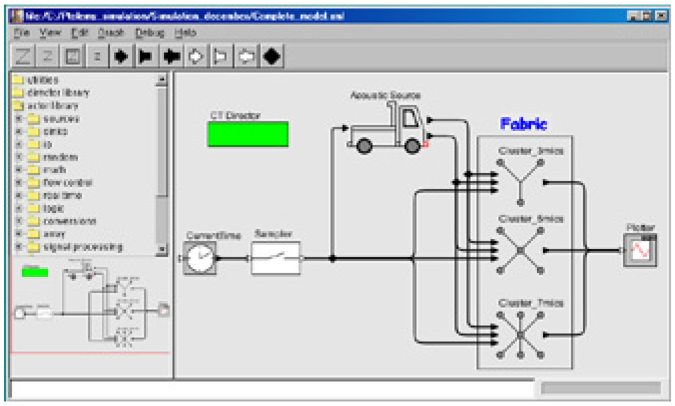


Figure 35. Ptolemy screen shot

VI.F. E-Textiles Conclusion and Future Challenges

This research demonstrated that e-textiles yield many advantages for use a large-scale sensor networks. Tangible results of this project are a series of e-textile prototypes culminating in a thirty-foot acoustic beam-forming array that demonstrated communication within the fabric and power-aware operation superior to that of purely wireless networks as well as a modeling and simulation framework for exploring e-textile design options.

During the course of our investigation we have identified the following challenges and opportunities:

- *Software Services*: We identified a set of software services common to a wide range of e-textile applications. These services include self-location, fault-tolerant location-based communication, time synchronization, and power-aware task assignment. The construction of portable software to provide these services would be beneficial to a wide array of applications.
- *Packaging:* Textiles pose a very different packaging challenge than do traditional PCBs. The density and number of pins on the majority of discrete components far exceeds that achievable in a cost-efficient fashion on a textile. E-textiles are not a substitute for printed circuit boards in the sense that they are not a good means for integrating discrete components that have many pins. That level of integration should take place at the packaging level (similar to the Infineon wearable MP3 player) and the resulting package should have a small number of pins at a pitch suitable for textiles.
- *Extensive communication network*: To take advantage of their large area, e-textiles will necessarily contain many components. To allow for fault-tolerance, this number of components is increased further. The challenge is to provide a network topology, under the restrictions imposed by the weaving process that allows for a large number of very simple nodes to communicate in a fault-tolerant, power-aware fashion. We have proposed a network for this task, but it has not been tested in a large-scale environment.
- *Uniforms*: As recognized by U. S. Army Soldier Systems Center (Natick), uniforms are an ideal application for e-textiles. Incorporation of an array of sensors into the fabric can allow for soldier health monitoring, monitoring for the presence of chem./bio threats, energy harvesting, improved antennas, and augmented awareness (using acoustic, ultrasonic, and/or video sensors in a 360 degree field around the soldier to identify potential threats). Wearable textiles impose new challenges based on the need to piece garments, account for user motion, allow for user comfort, and account for individual user differences.
- *Programming methodology and environment*: The state-of-the-art in e-textile programming is non-existent. Programs must be written at the level of each processing element embedded in the system. There is a need to raise the level of extraction to that of programming the textile rather than programming individual nodes.
- *Modeling and simulation*: Without modeling and simulation, it is not possible to explore even a fraction of the design space. This project has presented a framework for a modeling and simulation environment along with one instance of that framework. The community at large requires a well-

documented, expandable framework that also accounts for the fabric weave design, the construction of a textile from the fabric, human motion and individual differences as well as other factors.

- *New applications*: The range of applications is clearly vast, but some areas are immediately promising. There exists an immediate need for a textile (uniform and/or backpack) that is capable of storing power harvested from the environment; such a textile would greatly decrease soldier's dependence on a battery supply. There also exists an immediate need to incorporate a range of antennas into a single textile; discussions with Raytheon indicate a wide range of methods for exploiting e-textiles as a platform for antennas.
- *New materials integration*: There are a large number of new materials, many in fiber form factors, whose integration into e-textiles would be useful. Such fibers include energy storage, energy generation, and chemical sensing. Rather than making "one more thing for a soldier to carry" we should attempt to integrate these into the uniform the soldier already wears, tents, camouflage nets, and/or parachutes.

VII. Conclusions

The PAMA project has been able to demonstrate significant power savings in both software and hardware for power-aware multiprocessors. Demonstrations were provided in both software and hardware. The PAMA team has delivered a platform that supports power management and that has a rich programming environment. Algorithmic techniques were shown that can deliver energy savings of five orders of magnitude by trading accuracy for energy consumption and opportunities in hardware for power savings of up to 40 have been demonstrated. The three generations of hardware implementations of the PAMA architecture showed that power management techniques can be applied to a wide range of commercial hardware components with different trade-offs of performance, features, programmability, and power and energy savings. The PAMA project showed that significant power savings can be achieved by scaling the frequency and voltage of the processor, but that scaling the frequency and voltage of other components becomes important as processor power consumption is reduced through power management.

PAMA technology is being leveraged for the AMPS (Awareness and Management of Power for Space) as part of Phase 2 of DARPA's PAC/C program. Because the AMPS team is bringing its own applications to the program, the PAMA team has focused on developing platform development at the end of the PAMA program. PAMA technology is also being transitioned to the National Aeronautics and Space Administration (NASA), through their New Millennium Program. As we progress along the Moore's Law curve and performance increases at the expense of increased power consumption, we expect that the importance and acceptance of power management techniques will increase.

VIII. References

[1] Los Alamos National Laboratory Fast On-orbit Recoding of Transient Events Project (FORTÉ) http://www.FORTÉ.lanl.gov/nis-projects/FORTÉ/intro.html.

[2] Mitsubishi Microcomputers, M32000D4BFP-80 Data Book, http://www.mitsubishichips.com/data/datasheets/mcus/ mcupdf/ds/e32r80.pdf.

[3] Chandrakasan, A. P., Sheng, S., Brodersen, R. W. "Low-Power CMOS Digital Design," IEEE Journals of Solid-State Circuits, vol. 27, no. 4, pp 473-484, April 1992.

[4] Dey, J., Kurose, J., Towsley, D. "On-Line Scheduling Policies for a Class of IRIS (Increasing Reward with Increasing Service) Real-Time Tasks," IEEE Transactions on Computers, vol. 45, no. 7, July 1996.

[5] Aydin, H., Melhem, R., Mosse, D., Mejia-Alvarez, P. "Optimal Reward-Based Scheduling for Periodic Real-Time Tasks," IEEE Transactions on Computers, vol. 50, no. 2, February 2001.

[6] Kang, D., Crago, S.P., Suh, J. "A Fast Resource Synthesis Technique for Energy-Efficient Real-Time Systems," IEEE Real-Time Systems Symposium, Austin, Texas, December 2002.

[7] JTRACK, NASA http://liftoff.msfc.nasa.gov/realtime/JTrack/, 2001

[8] Kang, D., Suh, J., Crago, S.P. "An Optimal Voltage Synthesis Technique for a Power-Efficient Satellite Application," DAC (Design Automation Conference), New Orleans, LA, June 2002.

[9] Amdahl, G. M. "Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability," AFIPS Conference, Reston, VA, 1967.

[10] Hong, I., Qu, G., Pokonjak, M., Srivastava, M. B. "Synthetic Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," IEEE Real-Time Systems Symposium, December 1998.

[11] Suh, J., Kang, D., Crago, S.P.,"Dynamic Power Management of Multiprocessor Systems," Tenth International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS) in conjunction with IPDPS, Fort Lauderdale, FL, April 2002.

[12] Shriver, Patrick M., Gokhale, Maya B., et al. "A Power-Aware, Satellite-Based Parallel Signal Processing Scheme," Power Aware Computing. Kluwer Academic/Plenum Publishers, New York, Ch. 13, pp 243-259, 2002.

[13] Suh, J., Li, C., Crago, S.P., Parker, R. "A PIM-Based Multiprocessor System," International Parallel and Distributed Processing Symposium, San Francisco, CA, April 2001.

[14] Enemark, D. C., Shipley, M. E. "The FORTÉ Receiver and Sub-Band Triggering Unit," 8th Annual AIAA/Utah State University Conference on Small Satellites. 1994.

[15] Press, W., et al. Numerical Recipes in C, 2nd edition, Cambridge University Press., pp 699-705, 1992.

[16] Edmison, J., Jones, M., Nakad, Z., and Martin, T. "Using Piezoelectric Materials for Wearable Electronic Textiles," Sixth International Symposium on Wearable Computers (ISWC 2002), pp. 41-48, October 2002.

[17] Nakad, Z., Jones, M., and Martin, T. "Communications in Electronic Textile Systems," 2003 International Conference on Communications in Computing (CIC 2003), pp. 37-43, June 2003.

[18] Jones., M., Martin, T., Nakad, Z., Shenoy, R., Sheikh, T., Lehn, D., Edmison J. "Analyzing the Use of E-textiles to Improve Application Performance," IEEE Vehicular Technology Conference 2003, Symposium on Wireless Ad hoc, Sensor, and Wearable Networks (VTC 2003) (extended abstract), October 2003.

IX. Appendices

IX.A. PAMA API

1. int PAMA_Set_cpu_freq (int pid, float frequency);
   pid: process id

   frequency: CPU frequency (MHz)

   Set the CPU frequency of processor pid to frequency (unblocked version)
2. int PAMA_Set_cpu_mode (int pid, int mode, int standby_time);
   pid: the process id

   mode: cpu mode (include active, sleep, suspend)

   standby_time: the time(milliseconds/microseconds) to wait before cpu mode changed

   Set the CPU mode of processor pid to mode after standby_time (unblocked version)
3. int PAMA_Set_cpu_voltage (int pid, float voltage);
   pid: the process id

   voltage: the cpu voltage (V)

   Set the CPU voltage of processor pid to voltage
4. int PAMA_Set_CMbus_freq (int pid, float frequency);
   pid: the process id

   frequency: the CPU memory bus frequency (MHz)

   Set the CPU memory bus frequency of processor pid to frequency (unblocked version)
5. int PAMA_Set_CNbus_freq (int pid, float frequency);
   pid: the process id

   frequency: the CPU network bus frequency (MHz)

   Set the CPU network bus frequency of processor pid to frequency (unblocked version)
6. int PAMA_Set_net_freq (float frequency);
   frequency: the network frequency (MHz)

   Set the PAMA network frequency to frequency (blocked version)
7. int PAMA_Set_Dpath (int pid, int factor);
   pid: the process id

   factor: width of data path

   Set data path (unblocked version)
8. int PAMA_BSet_cpu_freq (int pid, float frequency);
   pid: the process id

   frequency: the CPU frequency (MHz)

   Set the CPU frequency of processor pid to frequency (blocked version)
9. int PAMA_BSet_cpu_mode (int pid, int mode, int standby_time);
   pid----the process id

   mode: CPU mode (include active, sleep, suspend)

   standby_time: the time (milliseconds/microseconds) to wait before CPU mode changed

   Set the CPU mode of processor pid to mode after standby_time (blocked version)

10. int PAMA_BSet_cpu_voltage (int pid, int mode, float voltage);
    pid: the process id

    voltage: the CPU voltage (V)

    Set the CPU voltage of processor pid to voltage (blocked version)

11. int PAMA_BSet_CMbus_freq (int pid, int factor);
    pid: the process id

    factor: the CPU memory bus frequency factor (MHz)

    Set the CPU memory bus frequency of processor pid to 1/factor of CPU clock speed (blocked version)

12. int PAMA_BSet_CNbus_freq (int pid, int factor);
    pid: the process id

    factor: the CPU network bus frequency factor

    Set the CPU network bus frequency of processor pid to 1/factor of CPU speed (blocked version)

13. int PAMA_BSet_Dpath (int pid, int factor);
    pid: the process id

    factor: width of data path

    Set data path (blocked version)

14. float PAMA_Get_cpu_freq (int pid);
    pid: the process id

    Return the CPU frequency of process pid

15. int PAMA_Get_cpu_mode (int pid);
    pid: the process id

    Return the CPU mode of process pid

IX.B. Acronyms

AMPS	Awareness and Management of Power for Space
API	Application Programming Interface
CMOS	Complimentary Metal Oxide Semi-conductor
COTS	Commercial Off-The-Shelf
DARPA	Defense Applied Research Projects Agency
DMA	Direct Memory Access
DRAM	Dynamic Random Access Memory
EDF	Earliest Deadline First
FFT	Fast Fourier Transform
FORTE	Fast On-orbit Recording of Transient Events
FPGA	Field Programmable Gate Array
GFLOPS	Giga FLoating-point Operations Per Second
JPL	Jet Propulsion Laboratory
LEOS	Low Earth Orbit Satellite
LLF	Least Laxity First
LMS	Least Mean Squares
MB	Megabytes
MF	Match Filter bank
MIPS	Million Instructions per Second
ML	Maximum Likelihood
MPI	Message Passing Interface
NASA	National Aeronautics and Space Administration
OS	Operating System
PAC/C	Power Aware Computing and Communication
PAMA	Power Aware Multiprocessor Architecture
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PIM	Processor In Memory
QL	Quick Look
RF	Radio Frequency
RM	Rate Monotonic
RM-H	Rate Monotonic Harmonic
SBIR	Small Business Innovation Research
SLIIC	System Level Intelligent Intensive Computing
ST	Software Trigger-box
TEC	Total Electron Content
VET	Variable Execution Time
VF	Variable Frequency
VHF	Very High Frequency
WPUF	Weighted Power Usage Function